

Microsoft®

Official Academic Course



Microsoft® .NET Framework 3.5 ASP.NET Application Development

Microsoft Certified Technology Specialist Exam 70-562

Microsoft® Official Academic Course

**Microsoft® .NET Framework
3.5, ASP.NET Application
Development, Exam 70–562**



Credits

EXECUTIVE EDITOR	John Kane
DIRECTOR OF SALES	Mitchell Beaton
EXECUTIVE MARKETING MANAGER	Chris Ruel
MICROSOFT SENIOR PRODUCT MANAGER	Merrick Van Dongen of Microsoft Learning
EDITORIAL PROGRAM ASSISTANT	Jennifer Lartz
PRODUCTION MANAGER	Micheline Frederick
PRODUCTION EDITOR	Kerry Weinstein
CREATIVE DIRECTOR	Harry Nolan
COVER DESIGNER	Jim O'Shea
TECHNOLOGY AND MEDIA	Tom Kulesa/Wendy Ashenberg

This book was set in Garamond by Aptara, Inc. and printed and bound by Bind Rite Graphics.
The cover was printed by Phoenix Color.

Copyright © 2011 by John Wiley & Sons, Inc. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc. 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030-5774, (201) 748-6011, fax (201) 748-6008. To order books or for customer service, please call 1-800-CALL WILEY (225-5945).

Microsoft, ActiveX, Excel, InfoPath, Microsoft Press, MSDN, OneNote, Outlook, PivotChart, PivotTable, PowerPoint, SharePoint, SQL Server, Visio, Windows, Windows Mobile, Windows Server, and Windows Vista are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other product and company names mentioned herein may be the trademarks of their respective owners.

The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

The book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, John Wiley & Sons, Inc., Microsoft Corporation, nor their resellers or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Evaluation copies are provided to qualified academics and professionals for review purposes only, for use in their courses during the next academic year. These copies are licensed and may not be sold or transferred to a third party. Upon completion of the review period, please return the evaluation copy to Wiley. Return instructions and a free of charge return shipping label are available at www.wiley.com/go/returnlabel. Outside of the United States, please contact your local representative.

ISBN 978-0-470-57810-0

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Foreword from the Publisher

Wiley's publishing vision for the Microsoft Official Academic Course series is to provide students and instructors with the skills and knowledge they need to use Microsoft technology effectively in all aspects of their personal and professional lives. Quality instruction is required to help both educators and students get the most from Microsoft's software tools and to become more productive. Thus our mission is to make our instructional programs trusted educational companions for life.

To accomplish this mission, Wiley and Microsoft have partnered to develop the highest quality educational programs for Information Workers, IT Professionals, and Developers. Materials created by this partnership carry the brand name "Microsoft Official Academic Course," assuring instructors and students alike that the content of these textbooks is fully endorsed by Microsoft, and that they provide the highest quality information and instruction on Microsoft products. The Microsoft Official Academic Course textbooks are "Official" in still one more way—they are the officially sanctioned courseware for Microsoft IT Academy members.

The Microsoft Official Academic Course series focuses on *workforce development*. These programs are aimed at those students seeking to enter the workforce, change jobs, or embark on new careers as information workers, IT professionals, and developers. Microsoft Official Academic Course programs address their needs by emphasizing authentic workplace scenarios with an abundance of projects, exercises, cases, and assessments.

The Microsoft Official Academic Courses are mapped to Microsoft's extensive research and job-task analysis, the same research and analysis used to create the Microsoft Certified Technology Specialist (MCTS) exam. The textbooks focus on real skills for real jobs. As students work through the projects and exercises in the textbooks they enhance their level of knowledge and their ability to apply the latest Microsoft technology to everyday tasks. These students also gain resume-building credentials that can assist them in finding a job, keeping their current job, or in furthering their education.

The concept of life-long learning is today an utmost necessity. Job roles, and even whole job categories, are changing so quickly that none of us can stay competitive and productive without continuously updating our skills and capabilities. The Microsoft Official Academic Course offerings, and their focus on Microsoft certification exam preparation, provide a means for people to acquire and effectively update their skills and knowledge. Wiley supports students in this endeavor through the development and distribution of these courses as Microsoft's official academic publisher.

Today educational publishing requires attention to providing quality print and robust electronic content. By integrating Microsoft Official Academic Course products, *WileyPLUS*, and Microsoft certifications, we are better able to deliver efficient learning solutions for students and teachers alike.

Bonnie Lieberman

General Manager and Senior Vice President

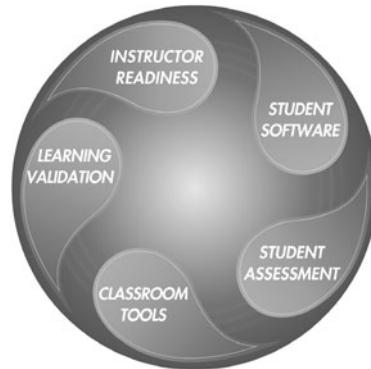
Preface

Welcome to the Microsoft Official Academic Course (MOAC) program for Microsoft .NET Framework 3.5, ASP.NET Application Development. MOAC represents the collaboration between Microsoft Learning and John Wiley & Sons, Inc. publishing company. Microsoft and Wiley teamed up to produce a series of textbooks that deliver compelling and innovative teaching solutions to instructors and superior learning experiences for students. Infused and informed by in-depth knowledge from the creators of Microsoft .NET Framework, and crafted by a publisher known worldwide for the pedagogical quality of its products, these textbooks maximize skills transfer in minimum time. Students are challenged to reach their potential by using their new technical skills as highly productive members of the workforce.

Because this knowledgebase comes directly from Microsoft, architect of the Microsoft .NET Framework and creator of the Microsoft Certified Technology Specialist and Microsoft Certified Professional exams (www.microsoft.com/learning/mcp/mcts), you are sure to receive the topical coverage that is most relevant to students' personal and professional success. Microsoft's direct participation not only assures you that MOAC textbook content is accurate and current; it also means that students will receive the best instruction possible to enable their success on certification exams and in the workplace.

■ The Microsoft Official Academic Course Program

The *Microsoft Official Academic Course* series is a complete program for instructors and institutions to prepare and deliver great courses on Microsoft software technologies. With MOAC, we recognize that, because of the rapid pace of change in the technology and curriculum developed by Microsoft, there is an ongoing set of needs beyond classroom instruction tools for an instructor to be ready to teach the course. The MOAC program endeavors to provide solutions for all these needs in a systematic manner in order to ensure a successful and rewarding course experience for both instructor and student—technical and curriculum training for instructor readiness with new software releases; the software itself for student use at home for building hands-on skills, assessment, and validation of skill development; and a great set of tools for delivering instruction in the classroom and lab. All are important to the smooth delivery of an interesting course on Microsoft software, and all are provided with the MOAC program. We think about the model below as a gauge for ensuring that we completely support you in your goal of teaching a great course. As you evaluate your instructional materials options, you may wish to use the model for comparison purposes with available products.



■ What Should I Know to Read This Book?

The audience for this book must have fundamental knowledge in any of the versions of the .NET Framework and Visual Studio. Since this book is based on C#, the book assumes that the readers are familiar with the following topics:

- Structure of a program, assemblies, GAC, and namespaces
- Scope of variables and access modifiers
- Operators, expressions, declarations, statements, generics
- Arrays
- Web site and page development
- Exception handling
- Introduction to debugging and state management
- Classes, methods, and interfaces
- Inheritance, abstraction, encapsulation, polymorphism
- Overloading and overriding
- Events, delegates, and event handlers
- Developing secured application for XSS and other attacks
- Compilation of code to assembly
- Basic networking concepts
- Implementation of SOAP headers
- Basics of XML
- Basics of JavaScript
- Installing IIS Web server
- FTP and other mechanisms for file transfer
- SQL statements and query
- Fundamentals of database access

■ What Is in This Book?

This book discusses the various objectives of the 70-562 exams in thirteen lessons.

Lesson 1: Introducing ASP.NET 3.5

This lesson provides an overview of application development using ASP.NET 3.5, such as:

- Entities in ASP.NET 3.5 application development
- Stages in ASP.NET application lifecycle
- ASP.NET application development structure
- Web site development
- Themes, skins, and master pages
- ASP.NET intrinsic objects
- ASP.NET compilation model

Lesson 2: Creating and Configuring Server Controls

This lesson discusses the server controls, such as:

- Standard Web server controls, such as Label, TextBox, Button, CheckBox, and RadioButton.

- Specialized Web server controls, such as Table, TableRow, TableCell, HotSpot, Calendar, Panel, Image, Wizard, and FileUpload.
- Data-bound controls, such as ListView, List, CompositeDataBoundControl and HierarchicalDataBoundControl, GridView, AdRotator, DetailsView, FormView, TreeView, Menu, DataGrid, DataList, Repeater, and DataPager controls.

Lesson 3: Working with User Controls and Web Parts

This lesson explains the methods to create and use various controls, such as:

- User controls
- Custom controls
- Composite controls
- Templated user controls
- WebParts controls

Lesson 4: Manipulating Data Using ADO.NET

This lesson describes the ways in which you can manipulate data in ASP.NET using connected and disconnected classes including the following:

- DataAdapter
- DataReader
- DataSet
- DataTable
- ObjectDataSource control
- SQLDataSource

Lesson 5: Using XML and LINQ

This lesson covers the techniques in using XML and LINQ such as:

- XML namespaces and classes
- XML documents
- LINQ to SQL entities
- Custom LINQ entity base classes
- Relational data

Lesson 6: Working with ASP.NET AJAX

This lesson deals with the fundamentals of AJAX, such as:

- AJAX architecture
- JSON
- AJAX controls, such as ScriptManager, ScriptManagerProxy, UpdatePanel, UpdateProgress, and Timer.
- AJAX events
- AJAX Web services

Lesson 7: Programming Web Applications

This lesson describes the techniques for programming web applications, including:

- Input validation
- Site navigation
- State management

Lesson 8: Working with Globalization, Localization, and Accessibility

This lesson explains how to improve accessibility of applications using mechanisms such as:

- Globalization
- Localization
- HTML layout best practices
- Public Accessibility Guidelines

Lesson 9: Enhancing Web Application Security and Performance

This lesson deals with various ways to enhance performance and security of applications, such as:

- Application data caching
- Output caching
- Authentication
- Authorization
- Role management
- Impersonation
- Membership
- Server controls

Lesson 10: Working with Services, Windows Communication Foundation, and ASP.NET Extensions

This lesson discusses various aspects of web applications, such as:

- XML Web services
- WCF Web services
- Silverlight
- MVC
- Dynamic data

Lesson 11: Working with Mobile Applications

This lesson covers the various aspects of mobile web application development, such as:

- Workflow of mobile web application
- Device-specific rendering
- Mobile Web forms
- Mobile controls, such as Container, Command, List, PhoneCall, StyleSheet, and RangeValidator
- Emulators
- Control adapters
- State management techniques, such as Cookieless sessions and View State

Lesson 12: Troubleshooting and Debugging Web Applications

This lesson focuses on the techniques for troubleshooting and debugging, such as:

- Page tracing
- Application tracing
- Trace statements
- Remote debugging

- Error handling
- Error pages
- Unhandled exceptions
- Custom and generic handlers

Lesson 13: Deploying and Publishing Web Applications

This lesson focuses on the techniques for deploying and publishing web applications, such as:

- Web Setup Project
- Deployment properties and conditions
- Copy Web tool
- Precompilation
- File System project
- HTTP project
- FTP project
- Health monitoring

Appendix A: Object-Oriented Programming

This appendix focuses on object-oriented programming language concepts, such as:

- Inheritance
- Encapsulation
- Polymorphism
- Abstraction

Illustrated Book Tour

■ Pedagogical Features

The MOAC textbook for .NET Framework 3.5, ASP.NET Application Development is designed to cover all the learning objectives for that MCTS exam, which is referred to as its “objective domain.” The Microsoft Certified Technology Specialist (MCTS) exam objectives are highlighted throughout the textbook. Many pedagogical features have been developed specifically for *Microsoft Official Academic Course* programs.

Presenting the extensive procedural information and technical concepts woven throughout the textbook raises challenges for the student and instructor alike. The Illustrated Book Tour that follows provides a guide to the rich features contributing to *Microsoft Official Academic Course* program’s pedagogical plan. Following is a list of key features in each lesson designed to prepare students for success on the certification exams and in the workplace:

- Each lesson begins with an **Lesson Skill Matrix**. More than a standard list of learning objectives, the Domain Matrix correlates each software skill covered in the lesson to the specific MCTS exam objective domain.
- Concise and frequent **Step-by-Step** instructions teach students new features and provide an opportunity for hands-on practice. Numbered steps give detailed step-by-step instructions to help students learn software skills. The steps also show results and screen images to match what students should see on their computer screens.
- **Illustrations:** Screen images provide visual feedback as students work through the exercises. The images reinforce key concepts, provide visual clues about the steps, and allow students to check their progress.
- **Key Terms:** Important technical vocabulary is listed at the beginning of the lesson. When these terms are used later in the lesson, they appear in bold italic type and are defined.
- Engaging point-of-use **Reader aids**, located throughout the lessons, tell students why this topic is relevant (*The Bottom Line*) or provide students with helpful hints (*Take Note*). Reader aids also provide additional relevant or background information that adds value to the lesson.
- **Certification Ready?** features throughout the text signal students where a specific certification objective is covered. They provide students with a chance to check their understanding of that particular MCTS exam objective and, if necessary, review the section of the lesson where it is covered.
- **Knowledge Assessments** provide progressively more challenging lesson-ending activities, including practice exercises and case scenarios.

■ Lesson Features

3

LESSON

Working with User Controls and Web Parts

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Working with user controls.	Load user controls dynamically.	2.2
Working with custom web controls.	Create and consume custom controls.	2.3
Using Web Parts controls.	Customize the layout and appearance of a web page.	7.1

KEY TERMS

composite control	templated user control	Web Parts
custom control	user control	zones

When developing web applications, there will come a time when you will want to develop your own controls that are better suited to the business requirements and that save you development time. You can do this by creating a standard template or by creating a control for the business data. For example, senior management might prefer its data in a graphical format instead of a tabular format because it helps them analyze the data and make better decisions. In contrast, team leaders might prefer to view data in a tabular format to get a detailed picture. ASP.NET provides many user controls and custom controls to achieve such goals. You can also combine functionalities of different server controls and provide new functionality.

In this lesson, you will learn how to create user controls, custom controls, and Web Parts controls.

Introducing User Controls

In the previous lesson, you learned about the built-in server controls that ASP.NET provides, including basic, specialized, and data-bound server controls. Again from these predefined controls, you can create your own controls to suit your requirements. Suppose you want to develop a web application in which you provide administrators and regular users with different user interface (UI) screens for logging on. Though the appearance of the screens might differ, the basic logic for logging on remains the same. Not only does this differentiate and ask users to provide their login credentials and, after authentication, it will display a welcome message. In this scenario, it is best to create a login control with standard login authentication logic that can be reused for all login UIs. When you create such a user control, the appearance and behavior of all the UIs based on it will be uniform and consistent. To modify the behavior and/or appearance of the UIs, you only need to modify the user control.

72

Warning Reader Aid

38 | Lesson 2

The **CheckedChange** event fires when the checked state of the control changes. As web application developers, you can add your code for execution in this event method. Figure 2-1 shows the various web server controls.

Figure 2-1
Web server controls

TAKENOTE

CREATE A SIMPLE WEB PAGE

WARNING Note that when you have learned about the basic server controls, let's create a web application that has a simple login page that asks users for their username and password. The page provides a Login button to log on and a Remember Me check box to remember the user details for subsequent logons.

To create this web page:

1. Add a new web form `Login.aspx` in your application.
2. From the Toolbox, drag and drop a Label and a TextBox control for entering the username. Set the properties as shown in this markup:


```
<asp:Label ID="lblUserName" runat="server" Text="User Name"></asp:Label>
<asp:TextBox ID="txtUserName" runat="server" height="22px" width="128px"></asp:TextBox>
```
3. This will display the user name label and the text box where users can type in their user name.
4. Add another Label and a TextBox control for entering the password. Once you set the properties, the following code is generated:


```
<asp:Label ID="lblPassword" runat="server" Text="Password"></asp:Label>
```

Lesson Skill Matrix

Key Terms

Certification Ready Alert

92 | Lesson 3

```
<td>Item Code:</td>
<td align="right"><%#Container.ItemCode %></td>
</tr>
</table>
</VendorTemplate>
</vc1:VendorInfoControl>
</div>
</form>
</body>
</html>
```

The control will be added to the .dll file. You can now reuse the .dll file.

TAKENOTE

Ensure that you are calling the `DataBind` method so that the vendor data is bound to the custom control, `VendorInfoControl`, when rendering the control.

UNDERSTANDING USER CONTROLS AND CUSTOM CONTROLS

In the previous sections, you learned about user and custom controls and how to create and use them. There are some differences between user controls and custom controls. You may be wondering when you should use a user control and when you should use a custom control; in other words, what are the basic differences between the two? Table 3-1 lists the differences between a user control and a custom control.

Table 3-1

User controls versus custom controls

USER CONTROLS	CUSTOM CONTROLS
Easy to create	Difficult to create
Provides limited support for UI design for developers who use visual design tools	Provides full support for developers who use visual design tools for UI design
Cannot be added to the toolbox	Can be added to the toolbox
Best suited for static layout	Best suited for dynamic layout
Presence of a local copy of the code is mandatory when used across applications	Local copies of the code are not mandatory; can be deployed on to the global assembly cache (GAC) for cross-application usage

Introducing Web Parts Controls

TAKENOTE

Suppose you want to develop a social networking Web site where registered users can change the look and feel of the site, customize their home page, or even rearrange the windows or controls that are visible to them. In addition, when the user signs out, you want to save all the customizations to show them again when they log on the next time. In order to do this, Web parts should be used. Web parts are a great feature that facilitates developers in achieving this flexibility by using Web Part controls. Web Part controls are widely used in SharePoint sites. Note that in a SharePoint site, you can very easily create Web Parts on your application using the wizard provided with minimal coding.

Using Web Parts

ASP.NET **Web Parts** are a collection of controls that help create Web sites in which users can modify the contents, behavior, and appearance of the web pages directly from the web

www.wiley.com/college/microsoft or
call the MOAC Toll-Free Number: 1+(888) 764-7001 (U.S. & Canada only)

136 | Lesson 4

Table 4-10
Arguments of the Fill event

PROPERTY	DESCRIPTION
Errors	Provides information about the exception that occurred.
DataTable	Represents the <code>DataTable</code> object on which the <code>FILL</code> method is executed and on which the error occurred.
Values	Represents an array of the row value where the error occurred while filling the data. The array value ordinal is directly correlated to that of the columns in the rows being added. <code>Value[0]</code> will correlate to the first column of the row.
Continue	Allows you to decide whether you need to throw an exception. If <code>Continue</code> is set to <code>True</code> , then an exception will be raised while the <code>FILL</code> method is used on the <code>DataSet</code> . If <code>Continue</code> is set to <code>False</code> then the <code>FILL</code> operation will be stopped.

CERTIFICATION READY
Using LINQ by using DataSet and DataReader objects.
3.2

Using Data Source Controls

ADO.NET has made great efforts to simplify and accelerate the development process, available, such as the data source controls. Access to the data source using the data source controls is not restricted to the database. These controls allow access with XML files and middle-tier business objects as well. You can easily connect to the data source to retrieve data and bind with other controls without massive coding.

THE BOTTOM LINE

ADO.NET has made great efforts to simplify and accelerate the development process, available, such as the data source controls. Access to the data source using the data source controls is not restricted to the database. These controls allow access with XML files and middle-tier business objects as well. You can easily connect to the data source to retrieve data and bind with other controls without massive coding.

Using Built-In Data Source Controls

The .NET Framework provides several built-in data source controls. These data source controls support data binding with data sources for different data-binding scenarios.

USING LINQDATASOURCE CONTROL

This control retrieves data from data source tables or from in-memory data tables, with the least amount of coding. While fetching data from the SQL server, you can configure this control to handle INSERT, UPDATE, and DELETE operations.

USING SOLIDDATASOURCE CONTROL

This control uses SQL commands for exchanging and modifying data. The `SqlDataSource` control can be used with Microsoft SQL Server, Oracle, OLEDB, and ODBC databases. The `DataReader` and `DataSource` objects hold the resultant data from the `SqlDataSource` control. `DataSet` helps cache the data, and it also supports sorting and filtering.

USING ENTITYDATASOURCE CONTROL

This control is based on the Entity Data Model (EDM), which is used for object-relation mapping by the .NET Framework. In addition, this control is also used for ADO.NET Data Services. Data binding takes place with the help of Entity-SQL (eSQL) querying language. The `EntityDataSource` control also supports queries created by the `ObjectQuery(T)` class.

USING ENTITYDATASOURCE CONTROL

This control interacts with the business objects or classes, which are basically the middle tier of a web application. In the case of the `IEnumerable` interface, the retrieved data

X Ref Reader Aid

Bottom Line Reader Aid

82 | Lesson 3

```
<%@ Register Src="~/CustomerControl.ascx" TagName="CustomerControl"
TagPrefix="uc1" %>
<!-- DOCTYPE PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    public void Page_Load()
    {
        DataBind();
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <uc1:CustomerControl ID="CustomerControl1" runat="server" CustomerID="1" CustomerName="Lamar" CustomerPhone="(501) 444-9843">
                <CustomerTemplate>
                    <h1>Customer Information</h1>
                    <span style="background-color:Lime">&nbsp;ID:&nbsp;</span>
                    <span style="background-color:Lime">&nbsp;Name:&nbsp;</span>
                    <div>Container.CustomerName %>
                    <div>Container.CustomerPhone %>
                </CustomerTemplate>
            </uc1:CustomerControl>
        </div>
    </form>
</body>
</html>
```

CERTIFICATION READY
Using user controls dynamically.
2.2

TAKENOTE You cannot view the interface of a templated user control during design time. Therefore, make sure that you run your web page to verify that it displays properly.

Working with Custom Web Controls

In this section, you will learn what custom controls are, how to create them, and how to use these controls in a web application.

Using XML and LINQ | 159

CREATE AN XMLDOCUMENT

To create an XMLDocument:

1. Add a new web form.
2. In the code-behind file, put the following code:

```
XmlNode doc = new XmlNode();
xDoc.AppendChild(doc.CreateDeclaration("1.0", "utf-8", null));
XmlElement root = xDoc.CreateElement("Microsoft");
xDoc.AppendChild(root);
XmlElement product = xDoc.CreateElement("Product");
XmlAttribute name = xDoc.CreateAttribute("Name");
name.Value = "Visual Studio";
XmlAttribute version = xDoc.CreateAttribute("Version");
version.Value = "9.0";
product.Attributes.Append(name);
product.Attributes.Append(version);
root.AppendChild(product);
xDoc.Save("C:\Product.xml");
```

This will create a file named Product.xml in the C:\ drive.

SKILL SUMMARY

In this lesson, you learned the advantages of XML and the various classes and namespaces in the .NET Framework that support XML. You also learned how to create a new XML document and how to parse an XML document using the `XDocument` class. In addition, you learned how to provide specific information in an XML document using different classes. You also learned about the `XnTextReader` and `XnTextWriter` classes that you use to read from and write data into an XML file. Then, you learned how to modify an XML document by adding new nodes. You also learned how to validate an XML document using DTD or schemas. In addition, you learned how to map objects to a relational database using LINQ to SQL. You also learned about the different ways of creating the OR map and the method to retrieve data from a database using LINQ to SQL. You also learned how to insert, update, and delete data using LINQ to SQL.

For the certification examination:

- Use and validate XML documents and read and write data to an XML file.
- Create LINQ and SQL entities and execute standard database commands with LINQ and SQL.

Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ is the base class for the `XnDataDocument` class. It is used to represent relational data and can expose the data as an object of `DataSet`.
2. The `CreateElement` and _____ methods are used to add nodes to the `XnDocument` object.
3. The _____ method searches an element in an XML document based on its ID.

ADO.NET Manipulating Data Using ADO.NET | 103

It is important to remember that ADO.NET is not a collection of ActiveX objects and that the letters A, D, and O in ADO.NET do not stand for ActiveX Objects. In fact, you probably already know that these letters mean nothing. Microsoft simply decided to continue with the abbreviation ADO so that it is not confusing for programmers shifting from ADO to ADO.NET.

ADO.NET utilizes the data providers of the .NET Framework for connecting to a database, executing commands, and retrieving results. The data manipulation and data access functionalities of ADO.NET are explicitly segregated into discrete components that can be used independently or together, like DataReaders, DataAdapters, and DataSets.

Before discussing connected and disconnected modes in detail, it is important to understand data providers.

Understanding Data Providers

Data providers enable you to connect to the data source, allow access to the data, and enable you to manipulate it.

Data providers are also known as data providers. Each provider is either processed directly or placed in an ADO.NET DataSet object. This DataSet object can be exposed to the user—in an ad-hoc manner or on-demand basis—without connecting to the data source, as long as the DataSet holds the latest data. This data can be combined with data from multiple sources, or it can be accessed from a remote location.

The .NET managed provider is the key element in the ADO.NET architecture. It is made up of a smaller set of interfaces and has simple data access architecture based on the .NET Framework 3.5. Figure 4-1 shows the ADO.NET architecture.

Figure 4-1
ADO.NET architecture

The .NET managed provider establishes the connection with the database to retrieve and manipulate data as required. It uses the help of the datatypes defined in the .NET Framework; the classes in the managed provider interact with the defined data source to return application-specific data to the application. Figure 4-1 shows the interconnection between the components of the managed provider.

92 | Lesson 3

CERTIFICATION READY?
Create and consume custom controls.
2.3

The control will be added to the .dll file. You can now reuse the .dll file.

TAKE NOTE
Ensure that you are calling the `.DataBind()` method so that the vendor data is bound to the custom control, `VendorInfoControl`, when rendering the control.

UNDERSTANDING USER CONTROLS AND CUSTOM CONTROLS
In the previous sections, you learned about user and custom controls and how to create and use them. There are some differences between user controls and custom controls. You may be wondering when you should use a user control and when you should use a custom control; in other words, what are the basic differences between the two? Table 3-1 lists the differences between a user control and a custom control.

Table 3-1
User controls versus custom controls

USER CONTROLS	CUSTOM CONTROLS
Easy to create	Difficult to create
Provides limited support for UI design for developers who use visual design tools	Provides full support for developers who use visual design tools for UI design
Cannot be added to the Toolbox	Can be added to the Toolbox
Best suited for static layout	Best suited for dynamic layout
Presence of a local copy of the code is mandatory when used across applications	Local copies of the code are not mandatory; can be deployed on to the global assembly cache (GAC) for cross-application usage

■ Introducing Web Parts Controls

Suppose you want to develop a social networking Web site where registered users can change the look and feel of the site, customize their home page, or even rearrange the windows or controls available to them. In addition, when the user signs in to the site, the user wants to save all the customizations to show them again when they log on the next time. In other words, a Web site should be highly configurable. ASP.NET facilitates developers in achieving this flexibility by using Web Part controls. Web Part controls are widely used in SharePoint sites. Note that in a SharePoint site, you can very easily create Web Parts on your application using the wizard provided with minimal coding.

Using Web Parts

ASP.NET Web Parts are a collection of controls that help create Web sites in which users can modify the contents, behavior, and appearance of the web pages directly from the web

More Information Reader Aid

88 | Lesson 3

[Designer("CustomControlSample.ImageControl"), ImageControlDesigner] public class ImageControl : WebControl

CREATING A COMPOSITE CONTROL

When a custom web server control contains other controls, it is called a *composite control*. A composite control is similar to the child control. However, it does not provide the designer interface and the `Toolbox`. To create a composite control, you create a class that is inherited from the `CompositeControl` class and then add the constituent controls to the `Controls` collection in the class you have created.

Note that a composite control is rendered as a tree of controls, each with its own life cycle and formation of a new application programming interface (API). Every child control handles its postback data and events on its own. This is very useful because you do not have to write any code for `PostBack` event handling.

If you are creating a composite control that is derived from the `CompositeControl` class and override the `CreateChildControls` method. The `CreateChildControls` method is available in the `Control` class that serves as a base class for `CompositeControl` class. This method should contain the code to instantiate the child controls and set their properties. You can use the `Panel` control to assign styles to your composite control.

`ReCreateChildControls` method of this class recreates the child controls in a control that is derived from `CompositeControl`.

CREATING A TEMPLATED CUSTOM CONTROL

Templated custom controls are used for the separation of control data and its presentation.

Similar to templated user controls, these controls do not provide any UI.

CREATE A TEMPLATED CUSTOM CONTROL

To create a templated custom control:

1. Create a ClassLibrary.dll project.
2. Add a System.Web.dll reference in the ClassLibrary.dll created in Step 1.
3. In the project, add a container class with public properties for the data that you want to access with the help of the container object, for example, `VendorInfoControl` as shown:

```
public class VendorInfoControl : Control, INamingContainer
{
    public int VendorID
    {
        get; set;
    }
    public string VendorName
    {
        get; set;
    }
    public int PurchaseOrderNumber
    {
        get; set;
    }
    public int ItemCode
    {
        get; set;
    }
}
```

4. Include the `System.Web.UI` namespace in the container class file.
5. Derive the `Container` class from the `System.Web.UI.Control` class and remember to inherit the `INamingContainer` interface.
6. Add a class `VendorInfoControl` to the project for the templated control.
7. Include the namespace `System.Web.UI` in the class that you created in Step 6.
8. Include the source code for the templated control to derive from the `System.Web.UI.Control` class and the `INamingContainer` interface in the templated control class.

Informative Diagrams

Take Note Reader Aid

MORE INFORMATION
For more information on the methods of the `Control` class, refer to the Control Class section on MSDN.

TAKE NOTE
If you are creating many composite controls with similar properties and/or methods, consider creating a base class for the composite control with common features.

CREATE A COMPOSITE CONTROL
When a custom web server control contains other controls, it is called a *composite control*. A composite control is similar to the child control. However, it does not provide the designer interface and the `Toolbox`. To create a composite control, you create a class that is inherited from the `CompositeControl` class and then add the constituent controls to the `Controls` collection in the class you have created.

Note that a composite control is rendered as a tree of controls, each with its own life cycle and formation of a new application programming interface (API). Every child control handles its postback data and events on its own. This is very useful because you do not have to write any code for `PostBack` event handling.

If you are creating a composite control that is derived from the `CompositeControl` class and override the `CreateChildControls` method. The `CreateChildControls` method is available in the `Control` class that serves as a base class for `CompositeControl` class. This method should contain the code to instantiate the child controls and set their properties. You can use the `Panel` control to assign styles to your composite control.

`ReCreateChildControls` method of this class recreates the child controls in a control that is derived from `CompositeControl`.

CREATE A TEMPLATED CUSTOM CONTROL
Templated custom controls are used for the separation of control data and its presentation.

Similar to templated user controls, these controls do not provide any UI.

CREATE A TEMPLATED CUSTOM CONTROL
To create a templated custom control:

1. Create a ClassLibrary.dll project.
2. Add a System.Web.dll reference in the ClassLibrary.dll created in Step 1.
3. In the project, add a container class with public properties for the data that you want to access with the help of the container object, for example, `VendorInfoControl` as shown:

```
public class VendorInfoControl : Control, INamingContainer
{
    public int VendorID
    {
        get; set;
    }
    public string VendorName
    {
        get; set;
    }
    public int PurchaseOrderNumber
    {
        get; set;
    }
    public int ItemCode
    {
        get; set;
    }
}
```

4. Include the `System.Web.UI` namespace in the container class file.
5. Derive the `Container` class from the `System.Web.UI.Control` class and remember to inherit the `INamingContainer` interface.
6. Add a class `VendorInfoControl` to the project for the templated control.
7. Include the namespace `System.Web.UI` in the class that you created in Step 6.
8. Include the source code for the templated control to derive from the `System.Web.UI.Control` class and the `INamingContainer` interface in the templated control class.

Easy-to-Read Tables

84 | Lesson 3

ADD A CUSTOM CONTROL TO A WEB PAGE

To add a custom control to a web page:

- Open the web page in the designer mode.
- From the Toolbox expand CustomerControlsSample Components. This is the namespace of your custom control or you may see the name of the web application.
- Select MyTextBox, and drag and drop this control on the web page. Then you can use the custom control exactly the same way you use standard controls such as Button or TextBox.

You can also add the custom control dynamically on your web page. To add the MyTextBox control dynamically on a web page, use the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace CustomControlSample
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            MyTextBox t = new MyTextBox();
            t.LabelCaption = "Please enter your data here: ";
            t.LabelWidth = 220;
            Form.Controls.Add(t);
        }
    }
}
```

The custom control is added to the web page as shown in Figure 3-2.

Figure 3-2
Control rendered in browser

Screen Images

.NET Working with ASP.NET AJAX | 185

Review Questions

- Which of the following controls help you update only a specific part of a web page?
 - Timer
 - UpdatePanel
 - ScriptManagerProxy
 - Button
- You have developed a web page where you have used UpdatePanels. You have placed UpdatePanel A and UpdatePanel B. Now, when you postback UpdatePanel B, which of the following will be updated?
 - Only UpdatePanel A will be updated.
 - Both UpdatePanel A and B will be updated.
 - Only UpdatePanel A will be updated.
 - None of the UpdatePanels will be updated.
- Which of the following components of AJAX enables serialization and deserialization?
 - Service proxy
 - JSON
 - Page methods
 - Web services

Review Questions

- What is partial page rendering?
- What are the component layers that contribute toward AJAX script libraries?

Case Scenarios

Scenario 6-1: Creating a Registration Page

You have to develop a Web site for your organization. The Web site requires a registration page that will display a set of common fields that a user first needs to fill in and a section with vendor and visitor specific fields. The user should be able to access the relevant fields by selecting the radio buttons Vendor and Visitor. The questions should be loaded dynamically without refreshing the page. How will you achieve this?

Scenario 6-2: Displaying Animations

Consider that you are developing a Web site to provide information about some products. You need to show a section of news with the product information in the home page of the Web site. The products will be displayed without any user interaction. However, you should ensure that it is possible to add or delete product information without much hassle later on. Therefore, you cannot use a gif animation file. How will you develop the Web site?

Workplace Ready

Developing Rich Web Sites

You need to develop a Web site that has a section to display the latest stocks and news update. These updates should refresh every three minutes. How will you design the Web site?

Skill Summary

.NET Using XML and LINQ | 159

CREATE AN XMLDOCUMENT

To create an XMLDocument:

- Add a new web form.
- In the code-behind file, put the following code:

```
Xmldoc = new XmlDocument();
xDoc.AppendChild(xDoc.CreateXmlDeclaration("1.0", "utf-8", null));
XmlElement root = xDoc.CreateElement("Microsoft");
xDoc.AppendChild(root);
XmlElement product = xDoc.CreateElement("Product");
XmlAttribute name = xDoc.CreateAttribute("Name");
name.Value = "Microsoft Visual Studio";
XmlElement version = xDoc.CreateAttribute("Version");
version.Value = "3.0";
product.Attributes.Append(name);
product.Attributes.Append(version);
root.AppendChild(product);
xDoc.Save("C:\\Product.xml");
```

This will create a file named Product.xml in the C:\\ drive.

SKILL SUMMARY

In this lesson, you learned the advantages of XML and the various classes and namespaces in the .NET Framework that support XML. You also learned how to create a new XML document and how to read and execute XML files using the XmlDocument class. You also learned how to search for specific information in an XML document using different classes. You also learned about the XmlNodeReader and XmlNodeWriter classes that you use to read from and write data into an XML file. Then, you learned how to modify an XML document by adding new nodes. You also learned how to validate an XML document using DTD or schemas. In addition, you learned about mapping objects to a relational database using LINQ to SQL. You also learned about the different ways of creating the OR map and the method to retrieve data from a database using LINQ to SQL. You also learned how to insert, update, and delete data using LINQ to SQL.

For the certification examination:

- Use and validate XML documents and read and write data to an XML file.
- Create LINQ and SQL entities and execute standard database commands with LINQ and SQL.

Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

- _____ is the base class for the XmlNode class. It is used to represent relational data and can expose the data as an object of DataSet.
- The CreateElement and _____ methods are used to add nodes to the XmlNode object.
- The _____ method searches an element in an XML document based on its ID.

Case Scenarios

Workplace Ready

Conventions and Features Used in This Book

This book uses particular fonts, symbols, and heading conventions to highlight important information or to call your attention to special steps. For more information about the features in each lesson, refer to the Illustrated Book Tour section.

CONVENTION	MEANING
 THE BOTTOM LINE	This feature provides a brief summary of the material to be covered in the section that follows.
CERTIFICATION READY?	This feature signals the point in the text where a specific certification objective is covered. It provides you with a chance to check your understanding of that particular MCTS objective and, if necessary, review the section of the lesson where it is covered.
TAKE NOTE*	Reader aids appear in shaded boxes found in your text. <i>Take Note</i> provides helpful hints related to particular tasks or topics.
	These notes provide pointers to information discussed elsewhere in the textbook or describe interesting features of Microsoft .NET Framework that are not directly addressed in the current topic or exercise.
A shared printer can be used by many individuals on a network.	Key terms appear in bold italic.

Instructor Support Program

The *Microsoft Official Academic Course* programs are accompanied by a rich array of resources that incorporate the extensive textbook visuals to form a pedagogically cohesive package. These resources provide all the materials instructors need to deploy and deliver their courses. Resources available online for download include:

- The **Instructor's Guide** contains Solutions to all the textbook exercises as well as chapter summaries and lecture notes. The Instructor's Guide and Syllabi for various term lengths are available from the Book Companion site (www.wiley.com/college/microsoft).
- The **Test Bank** contains hundreds of questions organized by lesson in multiple-choice, true-false, short answer, and essay formats and is available to download from the Instructor's Book Companion site (www.wiley.com/college/microsoft). A complete answer key is provided.
- **PowerPoint Presentations and Images.** A complete set of PowerPoint presentations is available on the Instructor's Book Companion site (www.wiley.com/college/microsoft) to enhance classroom presentations. Tailored to the text's topical coverage and Skills Matrix, these presentations are designed to convey key Microsoft .NET Framework concepts addressed in the text.

All figures from the text are on the Instructor's Book Companion site (www.wiley.com/college/microsoft). You can incorporate them into your PowerPoint presentations, or create your own overhead transparencies and handouts.

By using these visuals in class discussions, you can help focus students' attention on key elements of Windows Server and help them understand how to use it effectively in the workplace.

- When it comes to improving the classroom experience, there is no better source of ideas and inspiration than your fellow colleagues. The **Wiley Faculty Network** connects teachers with technology, facilitates the exchange of best practices, and helps to enhance instructional efficiency and effectiveness. Faculty Network activities include technology training and tutorials, virtual seminars, peer-to-peer exchanges of experiences and ideas, personal consulting, and sharing of resources. For details visit www.WhereFacultyConnect.com.



Important Web Addresses and Phone Numbers

To locate the Wiley Higher Education Rep in your area, go to the following Web address and click on the “Who’s My Rep?” link at the top of the page.

www.wiley.com/college

Or Call the MOAC Toll Free Number: 1 + (888) 764-7001 (U.S. & Canada only).

To learn more about becoming a Microsoft Certified Professional and exam availability, visit www.microsoft.com/learning/mcp.

Student Support Program

Book Companion Web Site (www.wiley.com/college/microsoft)

The students' book companion site for the MOAC series includes any resources, exercise files, and Web links that will be used in conjunction with this course.

Wiley Desktop Editions

Wiley MOAC Desktop Editions are innovative, electronic versions of printed textbooks. Students buy the desktop version for 50% off the U.S. price of the printed text, and get the added value of permanence and portability. Wiley Desktop Editions provide students with numerous additional benefits that are not available with other e-text solutions.

Wiley Desktop Editions are NOT subscriptions; students download the Wiley Desktop Edition to their computer desktops. Students own the content they buy to keep for as long as they want. Once a Wiley Desktop Edition is downloaded to the computer desktop, students have instant access to all of the content without being online. Students can also print out the sections they prefer to read in hard copy. Students also have access to fully integrated resources within their Wiley Desktop Edition. From highlighting their e-text to taking and sharing notes, students can easily personalize their Wiley Desktop Edition as they are reading or following along in class.

Microsoft Software

As an adopter of a MOAC textbook, your school's department is eligible for a free three-year membership to the MSDN Academic Alliance (MSDN AA). Through MSDN AA, full versions of Microsoft Visual Studio and Microsoft Expression are available for your use with this course. See your Wiley rep for details.

Preparing to Take the Microsoft Certified Technology Specialist (MCTS) Exam

The Microsoft Certified Technology Specialist (MCTS) certifications enable professionals to target specific technologies and to distinguish themselves by demonstrating in-depth knowledge and expertise in their specialized technologies. Microsoft Certified Technology Specialists are consistently capable of implementing, building, troubleshooting, and debugging a particular Microsoft Technology.

For organizations the new generation of Microsoft certifications provides better skills verification tools that help with assessing not only in-demand skills on Microsoft .NET Framework, but also the ability to quickly complete on-the-job tasks. Individuals will find it easier to identify and work toward the certification credential that meets their personal and professional goals.

To learn more about becoming a Microsoft Certified Professional and exam availability, visit www.microsoft.com/learning/mcp.

Microsoft Certified Technology Specialist

The new Microsoft Certified Technology Specialist (MCTS) credential highlights your skills using a specific Microsoft technology. You can demonstrate your abilities as an IT professional or developer with in-depth knowledge of the Microsoft technology that you use today or are planning to deploy.

The MCTS certifications enable professionals to target specific technologies and to distinguish themselves by demonstrating in-depth knowledge and expertise in their specialized technologies. Microsoft Certified Technology Specialists are consistently capable of implementing, building, troubleshooting, and debugging a particular Microsoft technology.

You can learn more about the MCTS program at www.microsoft.com/learning/mcp/mcts.

Microsoft Certified Professional Developer

The Microsoft Certified Professional Developer (MCPD) credential validates a comprehensive set of skills that are necessary to deploy, build, optimize, and operate applications successfully by using Microsoft Visual Studio and the Microsoft .NET Framework. This credential is designed to provide hiring managers with a strong indicator of potential job success.

MCPD certification will help you validate your skill and ability to develop applications by using Visual Studio 2008 and the Microsoft .NET Framework 3.5. Certification candidates should have two to three years of experience using the underlying technologies that are covered in the exam. The available certification paths include the following:

- ASP.NET Developer 3.5 for developers who build interactive, data-driven ASP.NET applications by using ASP.NET 3.5 for both intranet and Internet uses.
- Windows Developer 3.5 for developers who build rich client applications for the Windows Forms platform by using the Microsoft .NET Framework 3.5.
- Enterprise Application Developer 3.5 for developers who build distributed solutions that focus on ASP.NET and Windows Forms rich-client experiences.

You can learn more about the MCTS program at www.microsoft.com/learning/mcp/mcpd.

Preparing to Take an Exam

Unless you are a very experienced user, you will need to use a test preparation course to prepare to complete the test correctly and within the time allowed. The *Microsoft Official Academic Course* series is designed to prepare you with a strong knowledge of all exam topics, and with some additional review and practice on your own, you should feel confident in your ability to pass the appropriate exam.

After you decide which exam to take, review the list of objectives for the exam. You can easily identify tasks that are included in the objective list by locating the Lesson Skill Matrix at the start of each lesson and the Certification Ready sidebars in the margin of the lessons in this book.

To take the MCTS test, visit www.microsoft.com/learning/mcp to locate your nearest testing center. Then call the testing center directly to schedule your test. The amount of advance notice you should provide will vary for different testing centers, and it typically depends on the number of computers available at the testing center, the number of other testers who have already been scheduled for the day on which you want to take the test, and the number of times per week that the testing center offers MCTS testing. In general, you should call to schedule your test at least two weeks prior to the date on which you want to take the test.

When you arrive at the testing center, you might be asked for proof of identity. A driver's license or passport is an acceptable form of identification. If you do not have either of these items of documentation, call your testing center and ask what alternative forms of identification will be accepted. If you are retaking a test, bring your MCTS identification number, which will have been given to you when you previously took the test. If you have not prepaid or if your organization has not already arranged to make payment for you, you will need to pay the test-taking fee when you arrive.

Acknowledgments

MOAC Instructor Advisory Board

We thank our Instructor Advisory Board, an elite group of educators who has assisted us every step of the way in building these products. Advisory Board members have acted as our sounding board on key pedagogical and design decisions leading to the development of these compelling and innovative textbooks for future Information Workers. Their dedication to technology education is truly appreciated.



Charles DeSassure, Tarrant County College

Charles DeSassure is Department Chair and Instructor of Computer Science & Information Technology at Tarrant County College Southeast Campus, Arlington, Texas. He has had experience as a MIS Manager, system analyst, field technology analyst, LAN Administrator, microcomputer specialist, and public school teacher in South Carolina. DeSassure has worked in higher education for more than ten years and received the Excellence Award in Teaching from the National Institute for Staff and Organizational Development (NISOD). He currently serves on the Educational Testing Service (ETS) iSkills National Advisory Committee and chaired the Tarrant County College District Student Assessment Committee. He has written proposals and makes presentations at major educational conferences nationwide. DeSassure has served as a textbook reviewer for John Wiley & Sons and Prentice Hall. He teaches courses in information security, networking, distance learning, and computer literacy. DeSassure holds a master's degree in Computer Resources & Information Management from Webster University.



Kim Ehlert, Waukesha County Technical College

Kim Ehlert is the Microsoft Program Coordinator and a Network Specialist instructor at Waukesha County Technical College, teaching the full range of MCSE and networking courses for the past nine years. Prior to joining WCTC, Kim was a professor at the Milwaukee School of Engineering for five years where she oversaw the Novell Academic Education and the Microsoft IT Academy programs. She has a wide variety of industry experience including network design and management for Johnson Controls, local city fire departments, police departments, large church congregations, health departments, and accounting firms. Kim holds many industry certifications including MCDST, MCSE, Security+, Network+, Server+, MCT, and CNE.

Kim has a bachelor's degree in Information Systems and a master's degree in Business Administration from the University of Wisconsin Milwaukee. When she is not busy teaching, she enjoys spending time with her husband Gregg and their two children—Alex, and Courtney.



Penny Gudgeon, Corinthian Colleges, Inc.

Penny Gudgeon is the Program Manager for IT curriculum at Corinthian Colleges, Inc. Previously, she was responsible for computer programming and web curriculum for twenty-seven campuses in Corinthian's Canadian division, CDI College of Business, Technology and Health Care. Penny joined CDI College in 1997 as a computer programming instructor at one of the campuses outside of Toronto. Prior to joining CDI College, Penny taught productivity software at another Canadian college, the Academy of Learning, for four years. Penny has experience in helping students achieve their goals through various learning models from instructor-led to self-directed to online.

Before embarking on a career in education, Penny worked in the fields of advertising, marketing/sales, mechanical and electronic engineering technology, and computer programming. When not working from her home office or indulging her passion for lifelong learning, Penny likes to read mysteries, garden, and relax at home in Hamilton, Ontario, with her Shih-Tzu, Gracie.



Margaret Leary, Northern Virginia Community College

Margaret Leary is Professor of IST at Northern Virginia Community College, teaching Networking and Network Security Courses for the past ten years. She is the co-Principal Investigator on the CyberWATCH initiative, an NSF-funded regional consortium of higher education institutions and businesses working together to increase the number of network security personnel in the workforce. She also serves as a Senior Security Policy Manager and Research Analyst at Nortel Government Solutions and holds a CISSP certification.

Margaret holds a B.S.B.A. and MBA/Technology Management from the University of Phoenix, and is pursuing her Ph.D. in Organization and Management with an IT Specialization at Capella University. Her dissertation is titled “Quantifying the Discoverability of Identity Attributes in Internet-Based Public Records: Impact on Identity Theft and Knowledge-based Authentication.” She has several other published articles in various government and industry magazines, notably on identity management and network security.



Wen Liu, ITT Educational Services, Inc.

Wen Liu is Director of Corporate Curriculum Development at ITT Educational Services, Inc. He joined the ITT corporate headquarters in 1998 as a Senior Network Analyst to plan and deploy the corporate WAN infrastructure. A year later he assumed the position of Corporate Curriculum Manager supervising the curriculum development of all IT programs. After he was promoted to the current position three years ago, he continued to manage the curriculum research and development for all the programs offered in the School of Information Technology in addition to supervising the curriculum development in other areas (such as Schools of Drafting and Design and Schools of Electronics Technology). Prior to his employment with ITT Educational Services, Liu was a Telecommunications Analyst at the state government of Indiana working on the state backbone project that provided Internet and telecommunications services to the public users such as K-12 and higher education institutions, government agencies, libraries, and healthcare facilities.

Wen Liu has an M.A. in Student Personnel Administration in Higher Education and an M.S. in Information and Communications Sciences from Ball State University, Indiana. He used to be the director of special projects on the board of directors of the Indiana Telecommunications User Association, and used to serve on Course Technology's IT Advisory Board. He is currently a member of the IEEE and its Computer Society.



Jared Spencer, Westwood College Online

Jared Spencer has been the Lead Faculty for Networking at Westwood College Online since 2006. He began teaching in 2001 and has taught both on-ground and online for a variety of institutions, including Robert Morris University and Point Park University. In addition to his academic background, he has more than fifteen years of industry experience working for companies including the Thomson Corporation and IBM.

Jared has a master's degree in Internet Information Systems and is currently ABD and pursuing his doctorate in Information Systems at Nova Southeastern University. He has authored several papers that have been presented at conferences and appeared in publications such as the Journal of Internet Commerce and the Journal of Information Privacy and Security (JIPSC). He holds a number of industry certifications, including AIX (UNIX), A+, Network+, Security+, MCSA on Windows 2000, and MCSA on Windows 2003 Server.

We thank Dave Hammer at Butler Community College, Igor Belagorudsky, and Jeff Riley for their diligent review, and for providing invaluable feedback in the service of quality instructional materials.

Focus Group and Survey Participants

Finally, we thank the hundreds of instructors who participated in our focus groups and surveys to ensure that the Microsoft Official Academic Courses best met the needs of our customers.

Jean Aguilar, Mt. Hood Community College	Catherine Binder, Strayer University & Katharine Gibbs School—Philadelphia	Greg Clements, Midland Lutheran College
Konrad Akens, Zane State College	Terrel Blair, El Centro College	Dayna Coker, Southwestern Oklahoma State University—Sayre Campus
Michael Albers, University of Memphis	Ruth Blalock, Alamance Community College	Tamra Collins, Otero Junior College
Diana Anderson, Big Sandy Community & Technical College	Beverly Bohner, Reading Area Community College	Janet Conrey, Gavilan Community College
Phyllis Anderson, Delaware County Community College	Henry Bojack, Farmingdale State University	Carol Cornforth, West Virginia Northern Community College
Judith Andrews, Feather River College	Matthew Bowie, Luna Community College	Gary Cotton, American River College
Damon Antos, American River College	Julie Boyles, Portland Community College	Edie Cox, Chattahoochee Technical College
Bridget Archer, Oakton Community College	Karen Brandt, College of the Albemarle	Rollie Cox, Madison Area Technical College
Linda Arnold, Harrisburg Area Community College—Lebanon Campus	Stephen Brown, College of San Mateo	David Crawford, Northwestern Michigan College
Neha Arya, Fullerton College	Jared Bruckner, Southern Adventist University	J.K. Crowley, Victor Valley College
Mohammad Bajwa, Katharine Gibbs School—New York	Pam Brune, Chattanooga State Technical Community College	Rosalyn Culver, Washtenaw Community College
Virginia Baker, University of Alaska Fairbanks	Sue Buchholz, Georgia Perimeter College	Sharon Custer, Huntington University
Carla Bannick, Pima Community College	Roberta Buczyna, Edison College	Sandra Daniels, New River Community College
Rita Barkley, Northeast Alabama Community College	Angela Butler, Mississippi Gulf Coast Community College	Anila Das, Cedar Valley College
Elsa Barr, Central Community College—Hastings	Rebecca Byrd, Augusta Technical College	Brad Davis, Santa Rosa Junior College
Ronald W. Barry, Ventura County Community College District	Kristen Callahan, Mercer County Community College	Susan Davis, Green River Community College
Elizabeth Bastedo, Central Carolina Technical College	Judy Cameron, Spokane Community College	Mark Dawdy, Lincoln Land Community College
Karen Baston, Waubonsee Community College	Dianne Campbell, Athens Technical College	Jennifer Day, Sinclair Community College
Karen Bean, Blinn College	Gena Casas, Florida Community College at Jacksonville	Carol Deane, Eastern Idaho Technical College
Scott Beckstrand, Community College of Southern Nevada	Jesus Castrejon, Latin Technologies	Julie DeBuhr, Lewis-Clark State College
Paulette Bell, Santa Rosa Junior College	Gail Chambers, Southwest Tennessee Community College	Janis DeHaven, Central Community College
Liz Bennett, Southeast Technical Institute	Jacques Chansavang, Indiana University—Purdue University Fort Wayne	Drew Dekreon, University of Alaska—Anchorage
Nancy Bermea, Olympic College	Nancy Chapko, Milwaukee Area Technical College	Joy DePover, Central Lakes College
Lucy Betz, Milwaukee Area Technical College	Rebecca Chavez, Yavapai College	Salli DiBartolo, Brevard Community College
Meral Binbasioglu, Hofstra University	Sanjiv Chopra, Thomas Nelson Community College	Melissa Diegnau, Riverland Community College
		Al Dillard, Lansdale School of Business
		Marjorie Duffy, Cosumnes River College

- Sarah Dunn, Southwest Tennessee Community College
 Shahla Durany, Tarrant County College—South Campus
 Kay Durden, University of Tennessee at Martin
 Dineen Ebert, St. Louis Community College—Meramec
 Donna Ehrhart, State University of New York—Brockport
 Larry Elias, Montgomery County Community College
 Glenda Elser, New Mexico State University at Alamogordo
 Angela Evangelinos, Monroe County Community College
 Angie Evans, Ivy Tech Community College of Indiana
 Linda Farrington, Indian Hills Community College
 Dana Fladhammer, Phoenix College
 Richard Flores, Citrus College
 Connie Fox, Community and Technical College at Institute of Technology West Virginia University
 Wanda Freeman, Okefenokee Technical College
 Brenda Freeman, Augusta Technical College
 Susan Fry, Boise State University
 Roger Fulk, Wright State University—Lake Campus
 Sue Furnas, Collin County Community College District
 Sandy Gabel, Vernon College
 Laura Galvan, Fayetteville Technical Community College
 Candace Garrod, Red Rocks Community College
 Sherrie Geitgey, Northwest State Community College
 Chris Gerig, Chattahoochee Technical College
 Barb Gillespie, Cuyamaca College
 Jessica Gilmore, Highline Community College
 Pamela Gilmore, Reedley College
 Debbie Glinert, Queensborough Community College
 Steven Goldman, Polk Community College
 Bettie Goodman, C.S. Mott Community College
 Mike Grabill, Katharine Gibbs School—Philadelphia
 Francis Green, Penn State University
 Walter Griffin, Blinn College
 Fillmore Guinn, Odessa College
 Helen Haasch, Milwaukee Area Technical College
 John Habal, Ventura College
 Joy Haerens, Chaffey College
 Norman Hahn, Thomas Nelson Community College
 Kathy Hall, Alamance Community College
 Teri Harbachek, Boise State University
 Linda Harper, Richland Community College
 Maureen Harper, Indian Hills Community College
 Steve Harris, Katharine Gibbs School—New York
 Robyn Hart, Fresno City College
 Darien Hartman, Boise State University
 Gina Hatcher, Tacoma Community College
 Winona T. Hatcher, Aiken Technical College
 BJ Hathaway, Northeast Wisconsin Tech College
 Cynthia Hauki, West Hills College—Coalinga
 Mary L. Haynes, Wayne County Community College
 Marcie Hawkins, Zane State College
 Steve Hebrock, Ohio State University Agricultural Technical Institute
 Sue Heistand, Iowa Central Community College
 Heith Hennel, Valencia Community College
 Donna Hendricks, South Arkansas Community College
 Judy Hendrix, Dyersburg State Community College
 Gloria Hensel, Matanuska-Susitna College University of Alaska Anchorage
 Gwendolyn Hester, Richland College
 Tammarra Holmes, Laramie County Community College
 Dee Hobson, Richland College
 Keith Hoell, Katharine Gibbs School—New York
 Pashia Hogan, Northeast State Technical Community College
 Susan Hoggard, Tulsa Community College
 Kathleen Holliman, Wallace Community College Selma
 Chastity Honchul, Brown Mackie College/Wright State University
 Christie Hovey, Lincoln Land Community College
 Peggy Hughes, Allegany College of Maryland
 Sandra Hume, Chippewa Valley Technical College
 John Hutson, Aims Community College
 Celia Ing, Sacramento City College
 Joan Ivey, Lanier Technical College
 Barbara Jaffari, College of the Redwoods
 Penny Jakes, University of Montana College of Technology
 Eduardo Jaramillo, Peninsula College
 Barbara Jauken, Southeast Community College
 Susan Jennings, Stephen F. Austin State University
 Leslie Jernberg, Eastern Idaho Technical College
 Linda Johns, Georgia Perimeter College
 Brent Johnson, Okefenokee Technical College
 Mary Johnson, Mt. San Antonio College
 Shirley Johnson, Trinidad State Junior College—Valley Campus
 Sandra M. Jolley, Tarrant County College
 Teresa Jolly, South Georgia Technical College
 Dr. Deborah Jones, South Georgia Technical College
 Margie Jones, Central Virginia Community College
 Randall Jones, Marshall Community and Technical College
 Diane Karlsbraaten, Lake Region State College
 Teresa Keller, Ivy Tech Community College of Indiana
 Charles Kemnitz, Pennsylvania College of Technology
 Sandra Kinghorn, Ventura College

- Bill Klein, Katharine Gibbs School—Philadelphia
Bea Knaapen, Fresno City College
Kit Kofoed, Western Wyoming Community College
Maria Kolatis, County College of Morris
Barry Kolb, Ocean County College
Karen Kuralt, University of Arkansas at Little Rock
Belva-Carole Lamb, Rogue Community College
Betty Lambert, Des Moines Area Community College
Anita Lande, Cabrillo College
Junnae Landry, Pratt Community College
Karen Lankisch, UC Clermont
David Lanzilla, Central Florida Community College
Nora Laredo, Cerritos Community College
Jennifer Larrabee, Chippewa Valley Technical College
Debra Larson, Idaho State University
Barb Lave, Portland Community College
Audrey Lawrence, Tidewater Community College
Deborah Layton, Eastern Oklahoma State College
Larry LeBlanc, Owen Graduate School—Vanderbilt University
Philip Lee, Nashville State Community College
Michael Lehrfeld, Brevard Community College
Vasant Limaye, Southwest Collegiate Institute for the Deaf – Howard College
Anne C. Lewis, Edgecombe Community College
Stephen Linkin, Houston Community College
Peggy Linston, Athens Technical College
Hugh Lofton, Moultrie Technical College
Donna Lohn, Lakeland Community College
Jackie Lou, Lake Tahoe Community College
Donna Love, Gaston College
Curt Lynch, Ozarks Technical Community College
Sheilah Lynn, Florida Community College—Jacksonville
Pat R. Lyon, Tomball College
Bill Madden, Bergen Community College
Heather Madden, Delaware Technical & Community College
Donna Madsen, Kirkwood Community College
Jane Maringer-Cantu, Gavilan College
Suzanne Marks, Bellevue Community College
Carol Martin, Louisiana State University—Alexandria
Cheryl Martucci, Diablo Valley College
Roberta Marvel, Eastern Wyoming College
Tom Mason, Brookdale Community College
Mindy Mass, Santa Barbara City College
Dixie Massaro, Irvine Valley College
Rebekah May, Ashland Community & Technical College
Emma Mays-Reynolds, Dyersburg State Community College
Timothy Mayes, Metropolitan State College of Denver
Reggie McCarthy, Central Lakes College
Matt McCaskill, Brevard Community College
Kevin McFarlane, Front Range Community College
Donna McGill, Yuba Community College
Terri McKeever, Ozarks Technical Community College
Patricia McMahon, South Suburban College
Sally McMillin, Katharine Gibbs School—Philadelphia
Charles McNerney, Bergen Community College
Lisa Mears, Palm Beach Community College
Imran Mehmood, ITT Technical Institute—King of Prussia Campus
Virginia Melvin, Southwest Tennessee Community College
Jeanne Mercer, Texas State Technical College
Denise Merrell, Jefferson Community & Technical College
Catherine Merrikin, Pearl River Community College
Diane D. Mickey, Northern Virginia Community College
Darrelyn Miller, Grays Harbor College
Sue Mitchell, Calhoun Community College
Jacquie Moldenhauer, Front Range Community College
Linda Motonaga, Los Angeles City College
Sam Mryyan, Allen County Community College
Cindy Murphy, Southeastern Community College
Ryan Murphy, Sinclair Community College
Sharon E. Nastav, Johnson County Community College
Christine Naylor, Kent State University Ashtabula
Haji Nazarian, Seattle Central Community College
Nancy Noe, Linn-Benton Community College
Jennie Noriega, San Joaquin Delta College
Linda Nutter, Peninsula College
Thomas Omerza, Middle Bucks Institute of Technology
Edith Orozco, St. Philip's College
Dona Orr, Boise State University
Joanne Osgood, Chaffey College
Janice Owens, Kishwaukee College
Tatyana Pashnyak, Bainbridge College
John Partacz, College of DuPage
Tim Paul, Montana State University—Great Falls
Joseph Perez, South Texas College
Mike Peterson, Chemeketa Community College
Dr. Karen R. Petitto, West Virginia Wesleyan College
Terry Pierce, Onondaga Community College
Ashlee Pieris, Raritan Valley Community College
Jamie Pinchot, Thiel College
Michelle Poertner, Northwestern Michigan College
Betty Posta, University of Toledo
Deborah Powell, West Central Technical College
Mark Pranger, Rogers State University
Carolyn Rainey, Southeast Missouri State University
Linda Raskovich, Hibbing Community College

Leslie Ratliff, Griffin Technical College	Beth Sindt, Hawkeye Community College	Brad Vogt, Northeast Community College
Mar-Sue Ratzke, Rio Hondo Community College	Andrew Smith, Marian College	Cozell Wagner, Southeastern Community College
Roxy Reissen, Southeastern Community College	Brenda Smith, Southwest Tennessee Community College	Carolyn Walker, Tri-County Technical College
Silvio Reyes, Technical Career Institutes	Lynne Smith, State University of New York-Delhi	Sherry Walker, Tulsa Community College
Patricia Rishavy, Anoka Technical College	Rob Smith, Katharine Gibbs School—Philadelphia	Qi Wang, Tacoma Community College
Jean Robbins, Southeast Technical Institute	Tonya Smith, Arkansas State University—Mountain Home	Betty Wanielista, Valencia Community College
Carol Roberts, Eastern Maine Community College and University of Maine	Del Spencer – Trinity Valley Community College	Marge Warber, Lanier Technical College—Forsyth Campus
Teresa Roberts, Wilson Technical Community College	Jeri Spinner, Idaho State University	Marjorie Webster, Bergen Community College
Vicki Robertson, Southwest Tennessee Community College	Eric Stadnik, Santa Rosa Junior College	Linda Wenn, Central Community College
Betty Rogge, Ohio State Agricultural Technical Institute	Karen Stanton, Los Medanos College	Mark Westlund, Olympic College
Lynne Rusley, Missouri Southern State University	Meg Stoner, Santa Rosa Junior College	Carolyn Whited, Roane State Community College
Claude Russo, Brevard Community College	Beverly Stowers, Ivy Tech Community College of Indiana	Winona Whited, Richland College
Ginger Sabine, Northwestern Technical College	Marcia Stranix, Yuba College	Jerry Wilkerson, Scott Community College
Steven Sachs, Los Angeles Valley College	Kim Styles, Tri-County Technical College	Joel Willenbring, Fullerton College
Joanne Salas, Olympic College	Sylvia Summers, Tacoma Community College	Barbara Williams, WITC Superior
Lloyd Sandmann, Pima Community College—Desert Vista Campus	Beverly Swann, Delaware Technical & Community College	Charlotte Williams, Jones County Junior College
Beverly Santillo, Georgia Perimeter College	Ann Taff, Tulsa Community College	Bonnie Willy, Ivy Tech Community College of Indiana
Theresa Savarese, San Diego City College	Mike Theiss, University of Wisconsin—Marathon Campus	Diane Wilson, J. Sargeant Reynolds Community College
Sharolyn Sayers, Milwaukee Area Technical College	Romy Thiele, Cañada College	James Wolfe, Metropolitan Community College
Judith Scheeren, Westmoreland County Community College	Sharron Thompson, Portland Community College	Marjory Wooten, Lanier Technical College
Adolph Scheiwe, Joliet Junior College	Ingrid Thompson-Sellers, Georgia Perimeter College	Mark Yanko, Hocking College
Marilyn Schmid, Asheville-Buncombe Technical Community College	Barbara Tietsort, University of Cincinnati—Raymond Walters College	Alexis Yusov, Pace University
Janet Sebesy, Cuyahoga Community College	Denise Tillery, University of Nevada Las Vegas	Naeem Zaman, San Joaquin Delta College
Phyllis T. Shafer, Brookdale Community College	Susan Trebelhorn, Normandale Community College	Kathleen Zimmerman, Des Moines Area Community College
Ralph Shafer, Truckee Meadows Community College	Noel Trout, Santiago Canyon College	We also thank Lutz Ziob, Merrick Van Dongen, Jim LeValley, Bruce Curling, Joe Wilson, Rob Linsky, Jim Clark, Jim Palmeri, Scott Serna, Ben Watson, and David Bramble at Microsoft for their encouragement and support in making the Microsoft Official Academic Course programs the finest instructional materials for mastering the newest Microsoft technologies for both students and instructors.
Anne Marie Shanley, County College of Morris	Cheryl Turgeon, Asnuntuck Community College	
Shelia Shelton, Surry Community College	Steve Turner, Ventura College	
Marilyn Shepherd, Danville Area Community College	Sylvia Unwin, Bellevue Community College	
Susan Sinele, Aims Community College	Lilly Vigil, Colorado Mountain College	
	Sabrina Vincent, College of the Mainland	
	Mary Vitrano, Palm Beach Community College	

This page intentionally left blank

Brief Contents

Preface iv

- 1** Introducing ASP.NET 3.5 1
- 2** Creating and Configuring Server Controls 31
- 3** Working with User Controls and Web Parts 72
- 4** Manipulating Data Using ADO.NET 102
- 5** Using XML and LINQ 144
- 6** Working with ASP.NET AJAX 162
- 7** Programming Web Applications 186
- 8** Working with Globalization, Localization, and Accessibility 233
- 9** Enhancing Web Application Security and Performance 259
- 10** Working with Services, Windows Communication Foundation, and ASP.NET Extensions 302
- 11** Working with Mobile Applications 326
- 12** Troubleshooting and Debugging Web Applications 351
- 13** Deploying and Publishing Web Applications 375

Appendix A 388

Appendix B 396

Index 399

This page intentionally left blank

Contents

Lesson 1: Introducing ASP.NET 3.5 1

Objective Domain Matrix 1

Key Terms 1

Introducing ASP.NET 3.5 2

Working with Microsoft ASP.NET 2

Understanding Key ASP.NET Application Components 2

Using HTTP Methods 4

Understanding Life Cycle Stages in an ASP.NET Application 5

Working with Web Application Events and Methods 7

Understanding Application Development Structure 7

Working with Web Application Project Types 7

Understanding Assemblies 8

Creating and Customizing Web Site Layout and Appearance 9

Understanding Web Site and Web Application Projects 9

Designing Web Sites with Master Pages 12

Working with Intrinsic Objects in ASP.NET 14

Applying Skins and Themes to a Web Site 18

Understanding ASP.NET Page Structure 19

Understanding File Configuration and Compilation 20

Configuring Web Applications 20

Understanding the ASP.NET Compilation Model 22

Skill Summary 27

Knowledge Assessment 28

Case Scenarios 30

Workplace Ready 30

Lesson 2: Creating and Configuring Server Controls 31

Objective Domain Matrix 31

Key Terms 31

Introducing Web Pages and Controls 31

Understanding the Life Cycle of a Web Page and Its Controls 31

Differentiating between Server and HTML Controls 32

Using Standard Web Server Controls 35

Using the Label Control 35

Using the TextBox Control 35

Using the Button Control 36

Using the CheckBox Control 36

Using the RadioButton Control 37

Using Specialized Web Server Controls 41

Displaying Data in Tables 41

Displaying Images 43

Using the HotSpot Control 43

Using the Panel Control 44

Using the Wizard Control 44

Using the FileUpload Control 48

Using the MultiView Control 49

Creating Data-Bound Controls 49

Creating a Data-Binding Collection 49

Binding Data to Data-Bound Controls 50

Using the ListView Control 51

Understanding the DataSource Objects 51

Using the DataBinder Class 52

Using Template Controls 53

Using List Controls 54

Using Composite Data-Bound Controls and Hierarchical Data-Bound Controls 55

Using the GridView Control 55

Using the AdRotator Control 56

Using the DetailsView Control 56

Using the FormView Control 58

Using the TreeView Control 60

Using the Menu Control 61

Using the DataGrid Control 62

Using the DataList Control 62

Using the Repeater Control 64

Using the DataPager Control 66

Skill Summary 69

Knowledge Assessment 70

Case Scenarios 71

Workplace Ready 71

Lesson 3: Working with User Controls and Web Parts 72

Objective Domain Matrix 72

Key Terms 72

Introducing User Controls 72

Utilizing User Controls 73

Working with Custom Web Controls 82

- Creating a Custom Web Server Control 83
- Individualizing a Custom Web Server Control 86

Introducing Web Parts Controls 92

- Using Web Parts 92
- Understanding Web Parts Application Development 96
- Using WebPartManager and WebZones 96

Skill Summary 98**Knowledge Assessment 99****Case Scenarios 100****Workplace Ready 101****Lesson 4: Manipulating Data Using ADO.NET 102****Objective Domain Matrix 102****Key Terms 102****Understanding ADO.NET 102**

- Understanding Data Providers 103

Using ADO.NET Connected Classes 106

- Working with the Connection Objects 106
- Working with Command Objects 113
- Working with DataReader and DataAdapter Objects 115

Using ADO.NET Disconnected Classes 115

- Using the DataTable Object 115
- Manipulating Data 118
- Using the DataSet Object 121

Manipulating Data Using DataSet, DataAdapter, and DataReader Objects 123

- Retrieving Data Using DataReader 123
- Working with the DataAdapter Object 124
- Understanding DataAdapter Events 134

Using Data Source Controls 136

- Using Built-In Data Source Controls 136

Skill Summary 141**Knowledge Assessment 141****Case Scenarios 142****Workplace Ready 143****Lesson 5: Using XML and LINQ 144****Objective Domain Matrix 144****Key Terms 144****Introducing XML 144**

- Using XML Namespaces and Classes 145
- Working with XML Documents 147

Accessing Data with LINQ and SQL 154

- Creating LINQ and SQL Entities 154
- Mapping Objects to Relational Data 155
- Querying Data with LINQ to SQL 157
- Performing Insert, Update, and Delete Using LINQ to SQL 158

Skill Summary 159**Knowledge Assessment 159****Case Scenario 161****Workplace Ready 161****Lesson 6: Working with ASP.NET AJAX 162****Objective Domain Matrix 162****Key Terms 162****Introducing AJAX 162**

- Exploring AJAX 163
- Understanding How AJAX Works 163
- Understanding AJAX Architecture 164
- Understanding JSON 165
- Working with ASP.NET Server-Side AJAX Support 168
- Working with ASP.NET Client-Side AJAX Support 169

Using AJAX in ASP.NET Applications 169

- Working with AJAX Server-Side Controls 169
- Working with AJAX at Client Side 176
- Working with AJAX Events 178
- Understanding Web Services in AJAX 179

Skill Summary 183**Knowledge Assessment 184****Case Scenarios 185****Workplace Ready 185****Lesson 7: Programming Web Applications 186****Objective Domain Matrix 186****Key Terms 186****Introducing Input Validation 186**

- Implementing Input Validation 187
- Understanding Client-Side Validation Support 198

Performing Site Management 200

- Working with Site Maps 201
- Working with Navigation Controls 205
- Understanding URL Mapping 218

Implementing State Management	220
Understanding States and State Management	220
Skill Summary	230
Knowledge Assessment	230
Case Scenarios	231
Workplace Ready	232

Lesson 8: Working with Globalization, Localization, and Accessibility 233

Objective Domain Matrix	233
Key Terms	233
Introducing Globalization and Localization	233
Implementing Globalization and Localization	234
Localizing Web Applications Using Resource Files	242
Utilizing Best Practices for Implementing Globalization	244
Demonstrating Globalization and Localization	245
Configuring Accessibility	251
Understanding Accessibility Needs in a Web Application	251
Skill Summary	257
Knowledge Assessment	257
Case Scenario	258
Workplace Ready	258

Lesson 9: Enhancing Web Application Security and Performance 259

Objective Domain Matrix	259
Key Terms	259
Using Caching to Improve Performance	260
Understanding Output Caching	260
Understanding Fragment Caching	266
Understanding Application Data Caching	266
Securing Web Applications	272
Implementing Authentication	272
Understanding Server Controls for Login Functionality	278
Implementing Authorization	291
Skill Summary	299
Knowledge Assessment	299
Case Scenarios	301
Workplace Ready	301

Lesson 10: Working with Services, Windows Communication Foundation, and ASP.NET Extensions	302
---	-----

Objective Domain Matrix	302
Key Terms	302
Introducing ASP.NET Web Services	302
Creating and Using ASP.NET Web Services	303
Understanding the Windows Communication Foundation	311
Understanding the Need for WCF	311
Identifying WCF Elements	311
Creating and Using a WCF Service	312
Using a WCF Service	315
Hosting WCF Applications	318
Understanding ASP.NET Extensions	318
Using Dynamic Data	319
Using Model View Controller	319
Introducing Silverlight	320
Skill Summary	324
Knowledge Assessment	324
Case Scenario	325
Workplace Ready	325

Lesson 11: Working with Mobile Applications 326

Objective Domain Matrix	326
Key Terms	326
Introducing Mobile Applications	326
Developing Mobile Applications	327
Understanding Device-Specific Rendering	330
Viewing and Testing Mobile Web Applications	332
Understanding Mobile Web Forms	333
Introducing Mobile Controls	335
Understanding Types of Mobile Controls	335
Working with Mobile Controls	336
Using Control Adapters	346
Understanding State Management in ASP.NET Mobile	347
Skill Summary	349
Knowledge Assessment	349
Case Scenarios	350
Workplace Ready	350

Lesson 12: Troubleshooting
and Debugging Web
Applications 351

Objective Domain Matrix 351

Key Terms 351

Introducing Web Application
Troubleshooting 351

Introducing Tracing 352

Introducing Debugging and Error Logging 357

Debugging with Visual Studio 358

Debugging Deployment Issues 361

Debugging Remotely 362

Using Error Handling 362

Understanding Custom and Generic Handlers 367

Skill Summary 372

Knowledge Assessment 372

Case Scenarios 374

Workplace Ready 374

Lesson 13: Deploying and Publishing
Web Applications 375

Objective Domain Matrix 375

Key Terms 375

Deploying Web Applications 375

Using Web Setup Project 375

Working with Deployment Properties and Conditions 377

Using the Copy Web Tool 380

Using Web Applications Precompilation 381

Publishing Precompiled Web Applications 382

Publishing Web Applications 383

Using File System Project 383

Using HTTP Project 383

Using FTP Project 384

Monitoring Application Health 384

Skill Summary 386

Knowledge Assessment 386

Case Scenario 387

Workplace Ready 387

Appendix A 388

Appendix B 396

Index 399

Introducing ASP.NET 3.5

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Overview of ASP.NET 3.5.	Configure application pools.	1.6
Overview of ASP.NET 3.5.	Handle events and control page flow.	7.6
Overview of ASP.NET 3.5.	Configure projects, solutions, and reference assemblies.	1.3
Creating and customizing the layout and appearance of Web sites.	Work with ASP.NET intrinsic objects.	7.2
Creating and customizing the layout and appearance of Web sites.	Customize the layout and appearance of a Web page.	7.1
Overview of ASP.NET 3.5.	Configure application pools.	1.6
Creating and customizing the layout and appearance of Web sites.	Implement business objects and utility classes.	7.4
Understanding configuration and compilation of files.	Compile an application by using Visual Studio or command-line tools.	1.7

KEY TERMS

assembly
client
dynamic-link library (DLL)
global assembly cache (GAC)
HTTP request
HTTP response
Internet Information Server

Internet service application programming interface (ISAPI)
master pages
Multipurpose Internet Mail Extension (MIME)
users
web browser
web pages
web server

With the evolution of advanced technologies, the Internet has emerged as the preferred solution for everything, and the Internet is not restricted to work-related activities. You probably do a lot of personal tasks through the Internet, from booking your airline tickets to paying your household bills. Imagine that you intend to buy a product from an online shopping site. The first thing you do is access the Web site of the online shop and check out the products. Once you have decided on a product, you make the payment using the payment gateway, and

after a specified duration, the product is delivered to the address you specified. This is the kind of data exchange that happens on the Internet.

This increasing dependency on the Internet has resulted in a demand for new features to help you perform a wide range of activities. Development of ASP.NET 3.5 is one such step in the evolution of Internet technology. ASP.NET 3.5 makes web application development easier and more sophisticated. Examples of advanced web applications that have been developed using ASP.NET 3.5 include online banking, online trading, and mobile surfing.

In this lesson, you will learn about the features of ASP.NET 3.5 that help you develop web applications to meet the rising demand of the World Wide Web (WWW).

■ Introducing ASP.NET 3.5



THE BOTTOM LINE

Today, web applications are not just HTML pages; they perform far more complicated tasks with the help of new controls and features available in ASP.NET 3.5. To develop such sophisticated applications, it is important to know the key differentiators and features of ASP.NET 3.5.

Working with Microsoft ASP.NET

ASP stands for Active Server Pages. ASP.NET has emerged as the most technologically advanced, feature-rich, powerful platform for building distributed applications that can be accessed using Hypertext Transfer Protocol (HTTP).

The complete ASP.NET platform comes as part of the Microsoft.NET Framework. ASP.NET 3.5 is the latest version of the ASP.NET platform. This version of ASP.NET includes the following features:

- Asynchronous JavaScript and XML (AJAX) capabilities
- New server controls to support rich graphical layout
- New `ListView` data control that displays data and provides a highly customizable user interface (UI)
- Ability to extend authentication of server-based forms, role management, and profile services

ASP.NET utilizes HTTP commands to communicate between the browser and server. ASP.NET receives, processes, and responds to ***HTTP requests***.

ASP.NET provides an abstract programming model that allows you to expose the required classes and methods that a web application needs to the external world. ASP.NET can be treated as a compiled piece of code that is made of reusable and extendable components written in languages such as C#, VB.NET, Jscript.NET, and J#. ASP.NET simplifies programming for developers by providing a simple programming model, feature-rich class libraries, and a rich set of UI controls and by integrating well with the Integrated Development Environment (IDE) such as Visual Web Developer and Visual Studio.

Understanding Key ASP.NET Application Components

Besides understanding the advanced features of ASP.NET 3.5, it is also important to understand the role of the ***web server***, ***web browser***, and HTTP because these components are essential to complete the life cycle of a web application. The web browser initiates a request for a web page that is handled by a web server. The server processes the request and sends back the response with the help of HTTP, and the web browser displays the results.

WORKING WITH WEB SERVER

For a browser to display a web application, it should first be deployed on a web server. On Windows, ***Internet Information Server*** is the preferred web server for deployment of web applications.

Web servers:

- Handle requests for pages containing code to be executed at the server, along with static HTML files.
- Respond with results for the code executed.
- Store data across web page requests.

Most Web sites contain several ***web pages***. Therefore, the web server is kept connected to the browser for a specific period for further page requests.

WORKING WITH WEB BROWSER

A web browser is software that helps display information in a structured manner. It allows ***users*** to interact with components such as text, images, videos, music, games, and hyperlinks. Initially, HTML was used to develop web pages, which were platform independent and could be rendered in any operating system with no constraint on the window size. Each web page request to the web server resulted in the web browser clearing the browser screen and displaying a new web page.

More recently, advanced technologies allow web browsers to execute code using scripting languages, such as JavaScript. Today's web browsers also support plug-ins that enhance the user's experience. Asynchronous JavaScript and XML (AJAX) allow web browsers to communicate with the web servers without clearing the existing web pages from the browser window. Overall, these advanced technologies provide users with a more robust and enhanced browsing experience.

WORKING WITH INTERNET INFORMATION SERVICES (IIS)

Fundamentally, all web application environments work the same way, regardless of the hardware/software platform used. All systems on the Internet primarily use port 80 for communication. To monitor port 80 for incoming HTTP requests, specific server software is required. This software enables the servers to respond to the request in a meaningful manner. On the Microsoft platform, IIS acts as a gatekeeper—a watchdog that intercepts HTTP requests from port 80. IIS is the web server on a Windows platform.

IIS splits its directory space by creating virtual directories for each web application. Virtual directories are well-defined, manageable sections of working space in the IIS directory. For example, if a user tries to retrieve a resource from a web server using the uniform resource locator (URL) `http://www.mypublisher.com/books/technicalbooks.htm`, the Web site is identified on the web server with the help of a URL that contains the resource and directs the request through the Internet. The Domain Name System (DNS) resolver resolves the address of the web server. Once the server receives the call, it looks for the `technicalbooks.htm` resource in the directory-type entity named `books`. In this case, `books` refers to a virtual directory.

As the name suggests, virtual directories are not physical directories. They only hold one or more mappings to physical directories. Similarly, one physical directory can also map to several virtual directories. Moreover, the names of the physical and virtual directories need not be the same. When a virtual directory receives data requests, it directs them to the appropriate data sources. The retrieved data is then presented to the requesting application as if it all had been stored in one location. This ability to reach to disconnected repositories makes virtual directories ideal for consolidating data stored in a distributed environment. Each virtual directory supports specific configuration properties for security, error handling, redirections, and application isolation options. It also allows configuring IIS extension ***dynamic-link libraries (DLLs)*** using the ***Internet service application programming interface (ISAPI)***. To configure IIS, Microsoft Windows provides a GUI that can be accessed through the Control Panel.

WORKING WITH ISAPI DLLS

Running code in a DLL is fast. The ISAPI DLLs handle web requests. For the purpose of this book, it is enough for you to know how ISAPI DLLs handle HTTP requests.

The ISAPI DLLs define the entry point for handling normal HTTP requests. Among all the entry points, `HttpExtensionProc` is the most important method. All ISAPI extension DLLs implement this singular function when responding to HTTP requests, though each one may respond differently.

The `HttpExtensionProc` method takes a single parameter: an `EXTENSION_CONTROL_BLOCK` structure, which includes all the context of the request. On receiving a request, `HttpExtensionProc` acts as an entry point for receiving the request. IIS uses the structure of the `EXTENSION_CONTROL_BLOCK` to package the information and then pass the information to the ISAPI DLL through the `HttpExtensionProc` method. ISAPI uses the library heavily to take action on a request. ISAPI extension DLLs are also responsible for parsing the information and utilizing it to process the request. For example, when a *client* places a request for a customer or product lookup with parameters in a query string (which is an instance of the `EXTENSION_CONTROL_BLOCK` structure), the ISAPI extension DLL uses the query string parameters to create a database query that is specific to the site and passes it on to the database. After the database server processes the query and passes on the results, the ISAPI DLL streams the results back to the client.

Using HTTP Methods

HTTP has been present throughout the existence of Internet technologies. This text-based, request-response protocol defines how web browsers and web servers communicate with each other. In HTTP, a client sends a request to the server as a request method—Uniform Resource Identifier (URI)—protocol and a message over a connection. HTTP works on the application layer, transmission control protocol (TCP) works on the transport layer, and the Internet protocol (IP) works on the Internet layer to make this communication work. By default, TCP directs all requests/data to port 80.

HTTP methods can be considered safe or idempotent based on their property to generate side effects. Some important HTTP methods include:

- **HEAD:** Retrieves the metadata of a resource, which is cacheable, but does not return the actual resource. This method is a safe method.
- **OPTIONS:** Returns information about the communication options that can be identified by the request Uniform Resource Identifier (URI).
- **GET:** Allows the client to retrieve any information that is identified by the request URI in the form of entity. This method is also a safe method.
- **POST:** Creates a new, dynamically named resource. The action performed by POST depends on the request URI and the server. Data retrieved using this method is not cached.

There are other HTTP methods that you can use such as CONNECT, TRACE, DEBUG, and DELETE.

USING HTTP REQUEST

If you request a site by typing a URL in a browser, the browser uses the available Domain Name System (DNS) to translate the server name, which you provided with the URL, into an IP address. The browser then opens a socket and connects to port 80 at that address. For example, `http://www.mypublisher.com/default.aspx` can take the following form:

```
GET /default.aspx HTTP/1.1
Host: www.mypublisher.com
```

This request for a Web site uses the HTTP GET command. This command is passed along with the URL of the resource you are requesting and the version of the HTTP protocol.

Two commonly used HTTP commands are GET and POST. The GET command retrieves information identified by the requested URL. The POST command sends the content enclosed with the request to the server for processing.

MORE INFORMATION

To know more about the other HTTP methods, refer to the Method Definitions section on w3.org.

An HTTP request can contain several headers. An HTTP header comprises text that provides information about the browser request. In this example, the line beginning with `Host:` is an HTTP header. The headers that are usually found in an HTTP request include:

- **User-Agent:** Identifies the type of browser that created the request.
- **Connection:** Keeps a connection open or closed.
- **If-Modified-Since:** Validates cache at client side.

USING HTTP RESPONSE

An **HTTP response** is the information sent from server to the web browser based on a request. This information is wrapped in the `Response` object of ASP.NET. This response comprises the following components:

- **Status line:** This is formed from the protocol version of the message.
- **Exit code:** Contains the information whether the request was successful or produced an error. For example, exit code 200 means that the request was successfully processed.
- **Additional information:** This is information such as the page content type, length, and message body that follows the status line in the form of headers as shown in the following example:

```
HTTP/1.1 200 OK
Server: Microsoft-IIS/6.0
Content-Type: text/html
Content-Length: 55
<html><body><h1>ASP.NET 3.5 is cool!</h1></body></html>
```

In this example, the server output is enclosed between the HTML tags. The server inserts a blank line between the last header and the content of the HTTP response according to the protocol standard.

The content is displayed based on the browser and the **Multipurpose Internet Mail Extension (MIME)** type. Some common MIME types used in web applications are text, image, audio, and video.

MORE INFORMATION

For more information about the MIME types used in web applications, refer to "Handling MIME Types" in the Internet Explorer section on MSDN.

WORKING WITH HTTP TROUBLESHOOTING TOOLS

CERTIFICATION READY?
Configure application pools.
1.6

Some common tools that can help in troubleshooting HTTP errors are Network Sniffer and Telnet. The Sniffer tool can capture the HTTP messages that are being exchanged as packets between the web server and the browser. Telnet is a terminal emulator that sends and receives textual data on port 23.

Understanding Life Cycle Stages in an ASP.NET Application

An ASP.NET application undergoes the following stages to process a request. A better understanding of these states will enable you to write appropriate code for the different application stages to achieve the desired results.

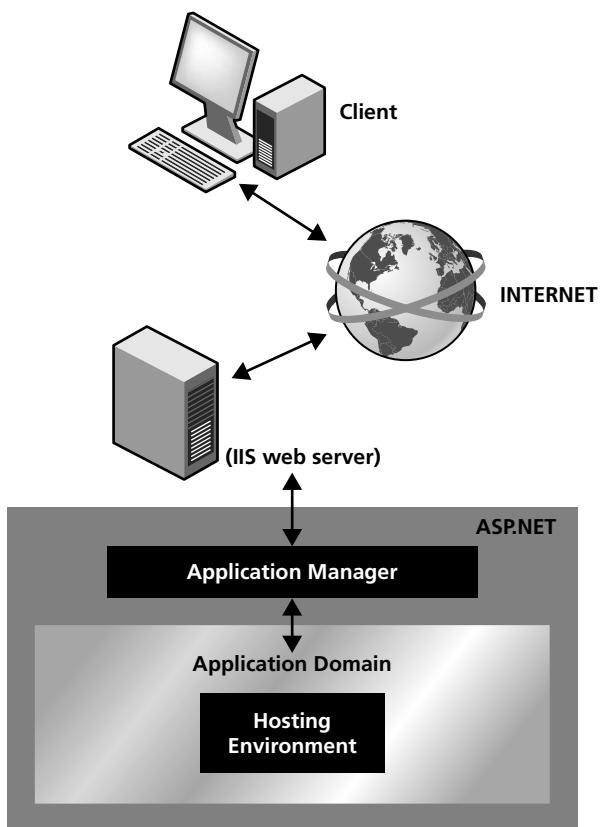
The key stages of an ASP.NET application's life cycle are:

1. **Client sends a request for an application resource:** An ASP.NET application's life cycle begins when a client requests an application resource. This request is sent by a browser to the web server, usually IIS. When a web server receives this request, it determines the appropriate ISAPI extension to handle the request based on the extension of the request. It then passes the request to the ISAPI extension for processing.

2. **ASP.NET receives the request:** When ASP.NET receives a request for any resource from a client for the first time, the ApplicationManager class creates an application domain. This Application domain separates the global variables and allows each application to be unloaded separately.
3. **ASP.NET creates core objects for each request:** Once the ApplicationManager creates the application domain, an instance of the HostingEnvironment object is created (see Figure 1-1). In addition, core objects, such as HttpContext, HttpRequest, and HttpResponse, are created and initialized.

Figure 1-1

ASP.NET application flow



4. **HttpApplication processes requests:** After creating the core objects, an instance of the HTTPApplication class is created, which starts the application. An instance of the global.asax file is also derived and instantiated.
5. **ASP.NET creates configured modules:** With the instantiation of the HttpApplication class, ASP.NET also creates configured modules.
6. **HttpApplication processes requests:** Next, the HttpApplication class executes events, such as BeginRequest, AuthenticateRequest, PostResolveRequest, and EndRequest, which are raised automatically. You can listen to these events using event listeners. The HTTPApplication class also validates requests and maps URLs configured in the web.config file in addition to executing the events.

TAKE NOTE*

ASP.NET is an ISAPI extension under the web server that handles file name extensions such as .aspx, .ascx, .ashx, and .asmx.

MORE INFORMATION

For information about the events executed by the **HTTPApplication** class and the sequence in which they are raised, refer to the "ASP.NET Page Life Cycle Overview" section on MSDN.

Working with Web Application Events and Methods

ASP.NET uses the `Application_event` naming convention to bind application events to handlers in the global.asax file.

The global.asax file is created at the root directory of an application and contains the code to handle application events, which are raised during the application's life cycle. ASP.NET compiles the global.asax file into a derived class of `HttpApplication`. This derived class is used to represent the application and processes only one request at a time. This makes it easy for you to handle application events. You can therefore store data specific to a request in the nonstatic members of the application class but should not need to lock the members while accessing them.

CERTIFICATION READY?

Handle events and control page flow.

7.6

MORE INFORMATION

To know more about the events and methods used during the life cycle of web applications, refer to the "ASP.NET Page Life Cycle Overview" section on MSDN.

Understanding Application Development Structure

Visual Studio is a container environment that integrates the functionality of multiple visual designers. For example, it provides designers or templates for building Windows applications, Web sites, and Windows Communication Foundation (WCF) services. All items that you require for application development, such as references, connectors to data sources, folders, and files, are grouped at two levels: projects and solutions.

Working with Web Application Project Types

There is no change in the project model of ASP.NET 3.5 from ASP.NET 2.0. Before creating a Web site in Visual Studio 2008, it is important to know the general architecture of Web sites. In Visual Studio 2008, web projects are called Web sites.

In Visual Studio 2008, you have several options for running and testing your Web site, based on its type. Some common types of Web sites are file, FTP, and local HTTP.

A file-based Web site consists of a folder, or folder structure, that contains all files for the Web site. It does not use or require IIS or Apache on the local machine. You can access this site only from a local computer, which helps secure the application. Additional advantages of using a file-based Web site are that you can enable multiple users to create and debug the Web site concurrently and also that no administrative rights are required to create or debug such a Web site. However, the downside of using a file-system-based site is that you cannot test it using application pooling or HTTP-based authentication.

You can use an FTP-based site when you need to work with files on a remote computer. The advantage of using a FTP-based site is that you can test it on the FTP server before deploying it.

An HTTP-based Web site should be used when you want to make the site accessible to other computers. You may configure the Web site with IIS or in a virtual directory. HTTP-based Web sites allow you to use HTTP authentication and application pooling. The downside is that you can debug such a Web site only if you have administrative rights to create and debug an IIS Web site.

In addition to the three types of Web sites discussed previously, you can also use a remote HTTP-based Web site when the application is hosted on a remote server. A remote Web site makes it possible for multiple users to access and work at the same time. In addition, you can test it on the server where you plan to deploy it. However, keep in mind that debugging a remote Web site can be a complex process.

WORKING WITH SOLUTION FILES

When you create a Web site, two files are created automatically: the solution file with a .sln extension and a binary solution user options file with a .suo extension. These files are located in My Documents\Visual Studio 2008\Projects folder. The .sln file contains information related to projects such as projects to be loaded, the source control system, and project dependencies.

Understanding Assemblies

You can think of an **assembly** as the primary unit that provides version control, deployment, and security in the .NET Framework. Assemblies are DLLs that can be .exe files as well. If you want to see where the .NET Framework assemblies are located on your machine, look at the local drive where you have Visual Studio installed. You will find the assemblies in the root folder. An assembly is known as a logical DLL because it can contain multiple .NET files. All the classes of the ASP.NET Framework are located in an assembly named System.Web.dll. In addition to local assemblies, you can also use external assemblies. External assemblies are those that are stored in a location other than the application folder.

There are two types of assemblies—private and shared. A private assembly can be used only by one application while shared assemblies can be used by any application located on the same server. You need to install all shared assemblies, including user-created shared assemblies that you would like to share across multiple applications to the **global assembly cache (GAC)**. When you compile user-created assemblies, those assemblies are placed in the bin subfolder inside the project folder.

By default, an ASP.NET application references the most common assemblies contained in GAC, which include:

- mscorlib.dll
- System.dll
- System.Configuration.dll
- System.Web.dll
- System.Data.dll
- System.Web.Services.dll
- System.Xml.dll
- System.Drawing.dll
- System.EnterpriseServices.dll
- System.Web.Mobile.dll

The .NET Framework 3.5 also references the following assemblies:

- System.Web.Extensions
- System.Xml.Linq
- System.Data.DataSetExtensions

You can use an assembly in your application by:

- Adding the assembly reference in the application.
- Writing code to import the namespace of the assemblies that you wish to use in your application.

You can add assembly references with the help of Visual Studio 2008. To do this:

1. In Visual Studio's Solution Explorer, right click the project or the bin folder and click Add Reference.
2. In the Add Reference dialog box, select the name of the assembly that you need.

For example, if you want to include printing functionality in the application that you are developing, you need to add the System.Printing.dll assembly. After adding this dll, the

TAKE NOTE*

You must supply the Version, Culture, and PublicKeyToken values associated with the assembly.

CERTIFICATION READY?

Configure projects, solutions, and reference assemblies.

1.3

TAKE NOTE*

You should avoid adding assemblies to GAC because using GAC defeats Xcopy deployment. This makes application back up and movement between servers difficult.

following code will be added within the <configuration><system.web><compilation> tag of the web configuration file (web.config):

```
<configuration>
  <system.web>
    <compilation debug="false">
      <assemblies>
        <add assembly="System.Printing, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Note that you can add only managed assemblies.

You can use the assemblies that are stored in GAC in your web application and App_Code components. App_Code components define the source code for your application. To use assemblies, you need to include a reference to the assembly in your web configuration file. The following code illustrates how you can add this reference using code:

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add assembly="MyLibrary, Version=0.0.0.0,
Culture=neutral, PublicKeyToken=250c66fc9dd31989"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

+ MORE INFORMATION

For more information on GAC and assemblies, refer to the "Global Assembly Cache" section on MSDN.

■ Creating and Customizing Web Site Layout and Appearance

**THE BOTTOM LINE**

Typically, a Web site is a collection of web pages, also called web forms in ASP.NET. You link these pages to enable users to navigate between the pages. To create a Web site, you create folders and add ASP.NET-specific files, which are processed by the .NET Framework.

Visual Studio refers to projects for developing Web sites as Web site projects. It provides templates that enable you to build new Web sites.

Understanding Web Site and Web Application Projects

Before you learn about how to create Web sites, let's first understand how Web site projects are different from web application projects. Selecting the New Web Site option allows you to create a file system Web site that stores pages and other files in a folder on your local computer.

A web application project type is an alternative to a Web site project. It uses a build model that creates a single assembly for the whole project. It provides some additional features that help you to manage applications in a flexible manner.

+ MORE INFORMATION

To identify the differences between Web sites and web applications, refer to the Web Application Projects Overview section on MSDN.

CREATING A WEB SITE

You have learned about the different components that play an important role in Web site development, such as server, browser, and HTTP protocol. Now, we will learn to create a new ASP.NET Web site and add pages to it.



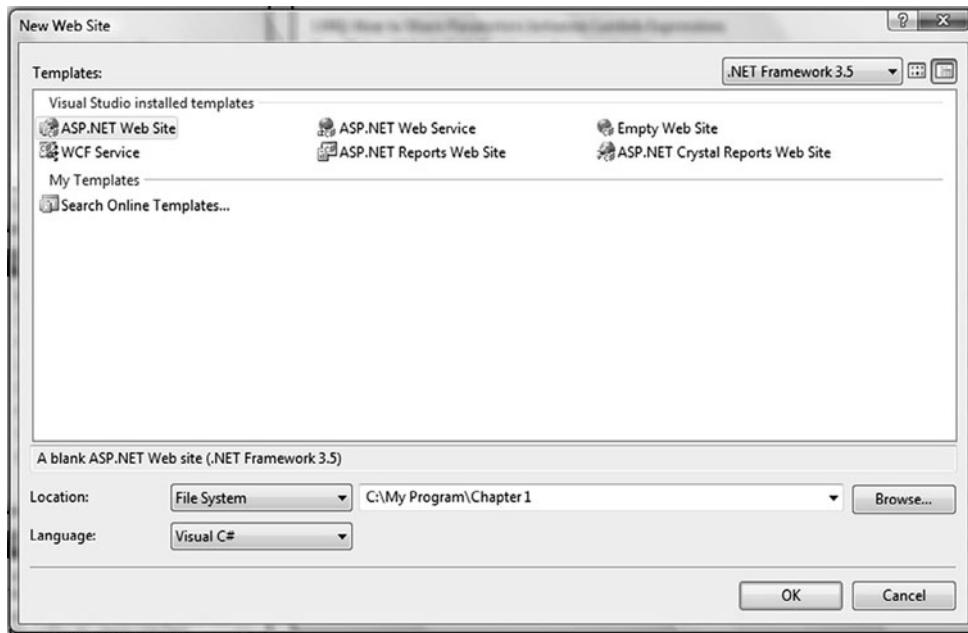
CREATE A WEB SITE AND ADD PAGES

To create a new Web site:

1. Go to File > New Web Site. You will see the window, as shown in Figure 1-2.

Figure 1-2

New Web Site window

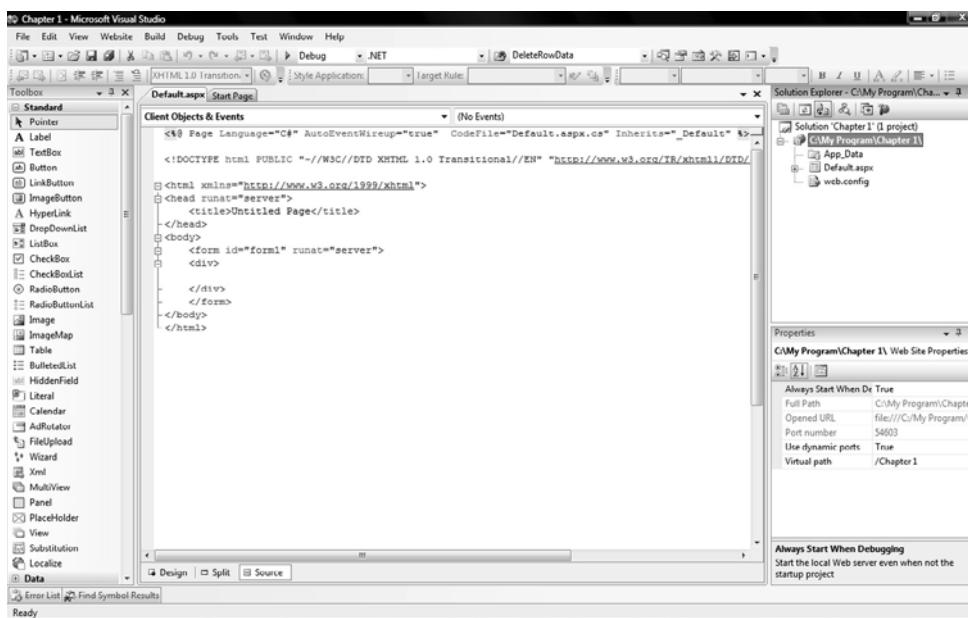


2. From the New Web Site window, select ASP.NET Web Site from the Templates section.
3. In the Location drop-down list, select File System and name the Web site Chapter 1.
4. Select the language as Visual C# from the Language drop-down list, and click the OK button.

The Visual Studio Integrated Development Environment (IDE) will be displayed as shown in Figure 1-3.

Figure 1-3

Visual Studio IDE

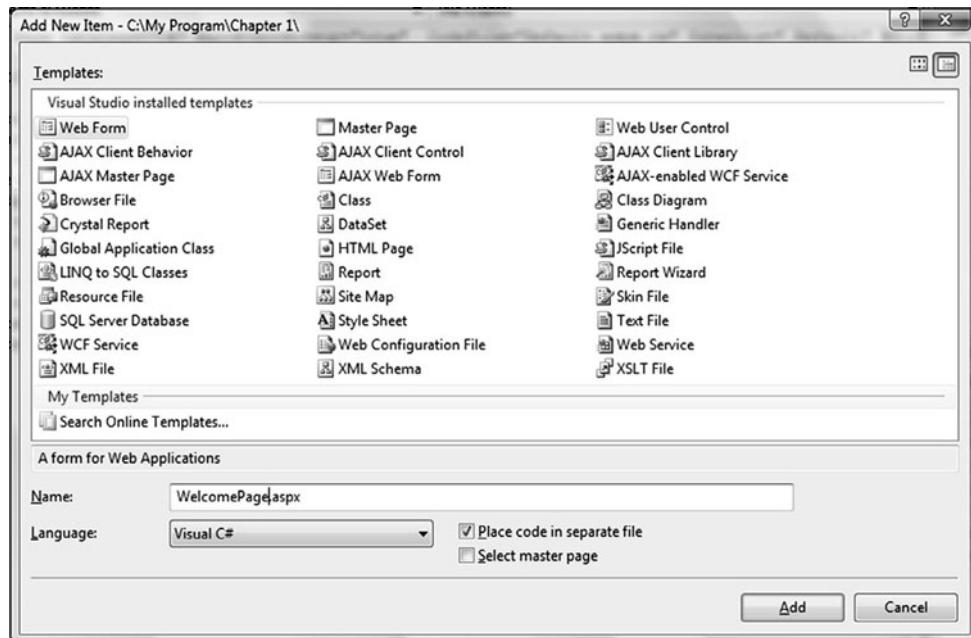


Now, we will add pages to the newly created Web site. To do this:

5. Go to the Solution Explorer, right click, and select Add New Item.
6. From the Add New Item window, select Web Form from the Templates list, as shown in Figure 1-4.

Figure 1-4

Selecting Web Form



7. Name the page WelcomePage.aspx.

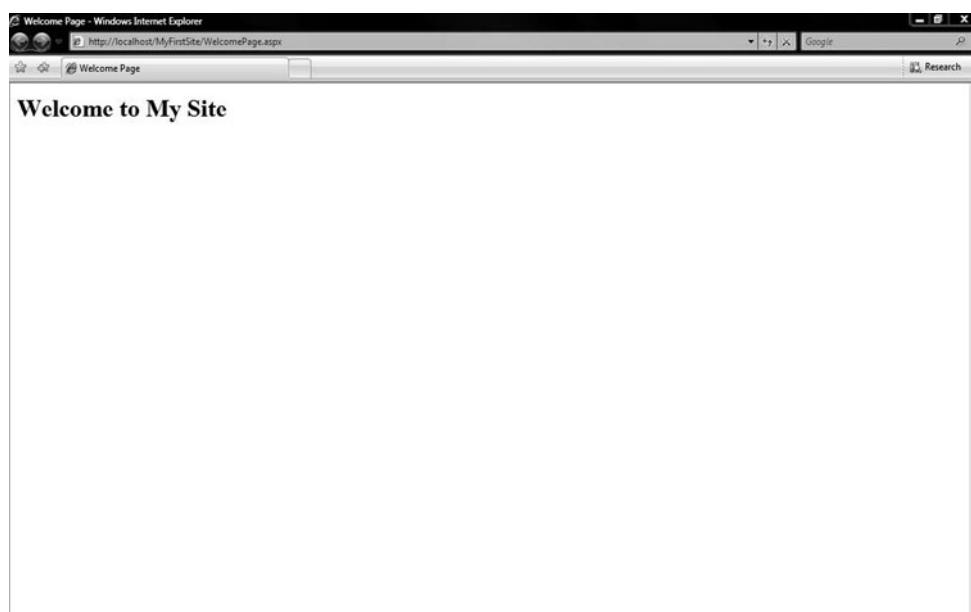
TAKE NOTE*

By default, the Language drop-down list will show Visual C# because you have set the language as Visual C# when creating the new Web site. If you did not specify the language, you can select it here and click the Add button.

8. To execute the page, right click WelcomePage.aspx in the Solution Explorer and select Set As Start Page.
9. Next, go to the main menu and select Debug > Start Debugging. The page will be displayed as shown in Figure 1-5. Alternatively, you can press F5 to execute the application.

Figure 1-5

New Web Site window



TAKE NOTE*

The master page without a `ContentPlaceHolder` control is technically correct. However, it doesn't behave like a template for multiple pages. `ContentPlaceHolders` can hold default content that is shown on a derived page unless overwritten.

Designing Web Sites with Master Pages

ASP.NET ***master pages*** help achieve uniformity throughout the application in terms of the appearance and layout. You can define the layout for the entire application in a single master page and merge the content pages with the master page for consistency. The master pages have two parts: the master page itself—for example, `mainMaster.master`—and one or more content pages. The master page is an ordinary ASP.NET page with a `.master` extension. However, unlike ordinary pages that have an `@Page` directive, the master page has an `@Master` directive at the top. The master page can have `ContentPlaceHolder` controls, which holds the default content shown on a derived page, unless it is overwritten.

Consider that you are designing a Web site where you want some images to appear in the header and footer. If you design the Web site without using master pages, then the code to insert the image would have to be replicated on all pages. Also, if you have to modify the images at a later point, you would have to update each page individually.

If you use master pages instead, you can simply add the code to the master page, and it is applied on all pages. Modifying the pages is easy.

The following code example shows you a master page:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MainMaster.master.cs" Inherits="MasterPage" %>

<!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></title>
    <asp:ContentPlaceholder id="cphHead" runat="server">
        </asp:ContentPlaceholder>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ContentPlaceholder id="cphMain" runat="server">
                </asp:ContentPlaceholder>
            </div>
        </form>
    </body>
</html>
```

A master page cannot be executed alone. It has to be bound to a page or an application. You can execute the master page like you executed the Web site in the earlier section. Note that you need to set the master page in the `@Page` directive for each page. A page bound to the master page automatically inherits all its contents, such as header and footer controls, and attaches custom markup and server controls to each defined placeholder. The content placeholder element is fully identified by its ID property and normally doesn't require other attributes.

The master page can be set at three levels:

- **Page level:** If this level is set, the content page can bind with a master page. The following line of code shows you how to set a master page at page level:

```
<%@ Page Language="C#" MasterPageFile="MainMaster.master"
AutoEventWireup="true" CodeFile="Home.aspx.cs" Inherits="_Default" %>
```

You can also set it programmatically on the page by including the following line of code:

```
this.MasterPageFile = "path"
```

- **Application level:** In the web.config file, by setting the `masterPageFile` property of the `pages` element to `MainMaster.master`, all the ASP.NET pages of the application will automatically bind to the master page, as shown:

```
<pages masterPageFile="MainMaster.master">
```

- **Folder level:** The implication is similar as application level, where the master page setting is done in the web.config file. However, this is done only if the file is present at the folder level. On setting, the master page will automatically be applied to the ASP.NET pages in that folder.

USING NESTED MASTER PAGES

Nested master pages contain a hierarchy of master pages in the form of a parent page and many child master pages. The child master page gives reference of the parent master page and inherits the parent's look and feel. This kind of structure is mostly useful for large Web sites that consist of more than one subsite. The subsites can define their own child master pages that reference the parent site master page. Only child master pages can add a reference to the parent master page as shown:

```
<%@ Master Language="C#"
MasterPageFile="ParentMasterPage.master" %>
```

USING @MASTER DIRECTIVE

The `@Master` directive helps the ASP.NET runtime distinguish the master page from other content pages. The master page is a special kind of ASP.NET control because it is compiled to a class that is derived from the `MasterPage` class and also inherits the `UserControl` class. Some of the important attributes of the `@Master` directive are listed in Table 1-1.

Table 1-1

Important attributes of the `@Master` directive

ATTRIBUTE	DESCRIPTION
<code>ClassName</code>	Provides a valid name for the class to be created that will render the master page. By default, the <code>ASP.simple_master</code> class is used. This class name should not include a namespace.
<code>CodeFile</code>	Specifies the URL of the file that contains the source code for the master page.
<code>Inherits</code>	Specifies any code-behind class derived from the master page for the master page to inherit.
<code>MasterPageFile</code>	Indicates the name of the master page file that is referenced by the <code>@Master</code> directive. You can create nested master pages by using this technique. A master page can reference another master page in the same way that any other page attaches to a master page.

Here is an example of the code file of the master page:

```
public partial class MasterPage: System.Web.UI.MasterPage
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
}
```

When you make a request for a content page, the server displays content along with the master page contents. However, the master page and the content page do need not be in the same folder. The `@Master` directive doesn't override attributes set at the `@Page` directive level. This means that the master page can have its language set to Visual Basic, while at the same time, one of the content pages may be using C#. The master page language setting never influences the language chosen at the content page level.

Working with Intrinsic Objects in ASP.NET

A request made to an ASP.NET application goes through several stages. First, the request is assigned to the `aspnet_isapi.dll`, which hands it over to the HTTP runtime pipeline. The entry point in the ASP.NET pipeline is the `HttpRuntime` class. A new instance of this class is created for each request. This instance monitors the overall execution and generates the response text for the browser. On instantiation, the `HttpRuntime` class performs a number of initialization tasks, the first of which is the creation of a wrapper to encapsulate all the HTTP-specific information available about the request. The newly created `HttpRuntime` object is then passed along the pipeline and is used by the various modules to access intrinsic worker objects, such as `Request`, `Response`, and `Server`.

In looking at the ASP.NET intrinsic objects and their properties, let's begin by examining the `ASP.NET Application` object.

USING THE `HTTPAPPLICATION` CLASS

All ASP.NET applications create an instance of the `HttpApplication` class, which provides a programmatic representation for the application. In addition, this class is the base class for all the applications that are defined by the user in the `Global.asax` file. The `HttpRuntime` class sets up the ASP.NET application object once the context for the request is created. This class also handles all HTTP requests directed to a particular virtual folder.

The virtual folder represents the ASP.NET runtime application. The virtual folder name works as a key to identify to which running application the request is made. However, the `Global.asax` file contains the setting and the code to respond to the events at application level raised by ASP.NET or the HTTP modules.

The `HttpApplication` object assigned to a request manages the request during its entire lifetime. No instance can be created until the response is completed. If no `HttpApplication` object is available, a new object is created and pooled with the existing ones.

The ASP.NET runtime creates an instance of `HttpApplication`, although the class is provided as a public constructor. A single instance can serve multiple requests, but can process only one request at a time. In case concurrent requests arrive for the same application, an additional instance of the class is created and added to the pool. Table 1-2 lists the properties of the `HttpApplication` class.

Table 1-2

Properties of the `HttpApplication` class

PROPERTY	DESCRIPTION
<code>Application</code>	An instance of the <code>HttpApplicationState</code> class that represents the global and shared state of the application. It is functionally equivalent to the ASP-intrinsic <code>Application</code> object.
<code>Context</code>	An instance of the <code>HttpContext</code> class that encapsulates all HTTP-specific information about the current request in a single object. Intrinsic objects, such as <code>Application</code> and <code>Request</code> , are also exposed as properties.
<code>Modules</code>	Gets the collection of modules that affect the current application.
<code>Request</code>	An instance of the <code>HttpRequest</code> class that represents the current HTTP request. It is functionally equivalent to the ASP-intrinsic <code>Request</code> object.
<code>Response</code>	An instance of the <code>HttpResponse</code> class that sends HTTP response data to the client. It is functionally equivalent to the ASP-intrinsic <code>Response</code> object.
<code>Server</code>	An instance of the <code>HttpServerUtility</code> class that provides helper methods for processing web requests. It is functionally equivalent to the ASP-intrinsic <code>Server</code> object.
<code>Session</code>	An instance of the <code>HttpSessionState</code> class that manages user-specific data. It is functionally equivalent to the ASP-intrinsic <code>Session</code> object.
<code>User</code>	An <code>IPrincipal</code> object that represents the user making the request.

The `HttpApplication` object determines the type of object that represents the resource being requested—typically, an ASP.NET page, a web service, or even a user control. The object then uses the appropriate handler factory to get an object that represents the requested resource. The factory either instantiates the class for the requested resource from an existing assembly or dynamically creates the assembly and then an instance of the class. A handler factory object is a class that implements the `IHttpHandlerFactory` interface and is responsible for returning an instance of a managed class (an HTTP handler) that can handle the HTTP request. An ASP.NET page is simply a handler object—an instance of a class that implements the `IHttpHandler` interface.

USING THE HTTPREQUEST CLASS

Suppose a browser is making a request for a page named `default.aspx`. To serve this request, the ASP.NET runtime should get a reference of the class `ASP.default.aspx` from the latest assemblies loaded in the AppDomain. If the reference is not found, a new reference is created. Next, the runtime environment invokes `ASP.default.aspx` class with the help of the `IHttpHandler` interface, which is implemented by the root `Page` class. After HTTP runtime obtains an instance of the class that represents the requested resource, the public `ProcessRequest` method is invoked, and it initiates the process that culminates in the generation of the final response for the browser. You have already seen that the life cycle of an ASP page is a collection of steps and events that execute and trigger the call to the `ProcessRequest` method. The ASP.NET worker process passes all the incoming HTTP requests to the HTTP pipeline regardless of whether they are `w3wp.exe` or `aspnet_wp.exe`.

TAKE NOTE *

`W3wp.exe` and `asp_wp.exe` are worker processes that help in allocating resources to applications. `W3wp.exe` is the worker process of IIS6.0, and `aspnet_wp.exe` is the worker process of IIS5.0. The `asp_wp.exe` process of IIS5.0 runs under the default account “`ASPNET`” unlike `w3wp.exe` that runs under “`NetworkService`” of IIS 6.0.

Incoming web requests, which contain the HTTP headers, query strings, form input fields, paths, and URL information, are organized in a group by the `HttpRequest` object. This enables easy and effective programming. The `HttpRequest` object is populated as soon as ASP.NET begins working on a web request and is made available through the `Request` property of the `HttpContext` object. An `HttpRequest` object exposes a fair number of properties and is one of the objects that has been significantly enriched in the transition from ASP to ASP.NET.

The properties of the `HttpRequest` class are categorized based on the type of information they contain. These groups are request type, client data, and connection.

Table 1-3 lists some important properties of the `HttpRequest` class.

Table 1-3

Properties of the `HttpRequest` class

PROPERTY	CATEGORY	DESCRIPTION
<code>AcceptTypes</code>	Client data	Gets an array of strings denoting the list of MIME types supported by the client for a specified request.
<code>AnonymousID</code>	Client data	Indicates the ID of the anonymous user, if any. The identity refers to the string generated by the <code>anonymousIdentification</code> module and has nothing to do with the identity of the IIS anonymous user. This property is not available in ASP.NET 1.x.
<code>Browser</code>	Client data	Gets an <code>HttpBrowserCapabilities</code> object that contains information about the capabilities of the client's browser.

(continued)

Table 1-3 (continued)

PROPERTY	CATEGORY	DESCRIPTION
ContentEncoding	Client data	Gets or sets an Encoding object that represents the client's character set. If specified, this property overrides the ASP.NET default encoding.
ContentLength	Client data	Gets length of the content sent by the client (in bytes).
ContentType	Request type	Gets or sets the MIME content type of the incoming request.
CurrentExecutionFilePath	Request type	Gets the current virtual path of the request even when the client is redirected to another page via <code>Execute</code> or <code>Transfer</code> methods. The <code>FilePath</code> property, on the other hand, always returns the path to the originally requested page.
FilePath	Request type	Gets the virtual path of the current request. The path doesn't change in case of server-side page redirections.
HttpMethod	Request type	Gets a string that denotes the HTTP method used for the request. This property can hold the values GET, POST, and HEAD.
RequestType	Request type	Gets or sets a string that denotes the HTTP command used to issue the request. Values can be GET or POST.
TotalBytes	Request type	Gets the total number of bytes in the input stream. This property differs from <code>ContentLength</code> in that it also includes headers.
UserAgent	Client data	Gets a string that identifies the browser. This property gets the raw content from the user agent header.

USING THE `HTTPCONTEXT` CLASS

The `HttpContext` class is instantiated by the `HttpRuntime` object when the request processing mechanism is set up. During the various stages in the execution of a request, an object gets passed from one class to another. This object is called the `HttpContext` object. The object encapsulates all the information available about an individual HTTP request that is waiting to be processed. You can use `HttpContext.Current` outside of the place where the `HttpContext` instance is passed, such as a class or request. Next, the object is passed through various stages of the request's lifetime.

Some important methods of the `HttpContext` class are `AddError`, `ClearError`, `GetConfig`, `ToString`, and `GetType`.

USING THE `HTTPRUNTIME` CLASS

The `HttpRuntime` class is the entry point after the page request is stacked in the pipeline of objects. This class enables you to configure the properties of the HTTP Runtime, such as the number of threads maintained in the thread pool. At the beginning, the ASP.NET worker process activates the HTTP pipeline and instantiates a new instance of the `HTTPRuntime` class. Then, the worker process calls its `ProcessRequest` method for each incoming request. This object processes the original HTTP request and generates some browser readable code at the end of the chain.

TAKE NOTE*

Despite the name, `HttpRuntime.ProcessRequest` has nothing to do with the `IHttpHandler` interface.

The `HttpRuntime` class contains many private and internal methods and only four public static methods: `Close`, `ProcessRequest`, `GetNamedPermissionSet`, and `UnloadAppDomain`. Table 1-4 discusses the public methods in detail.

Table 1-4

Methods of the
HttpRuntime class

METHOD	DESCRIPTION
Close	Clears all items from the ASP.NET cache and then terminates the web application. You do not usually use this method for normal ASP.NET request processing.
ProcessRequest	Drives all processing of requests in an ASP.NET web application.
UnloadAppDomain	Terminates the current ASP.NET application. The web application will restart when another request is received for it.
GetNamedPermissionSet	Returns the permissions assigned to code groups.

MORE INFORMATION

In addition to these methods, the **HttpRuntime** class contains other methods. To learn more about the other methods, refer to the “**HttpRuntime Members**” section on MSDN.

You should use the methods discussed in the table only when absolutely necessary. The **Close** method is only useful for hosting ASP.NET in a custom application. The **UploadAppDomain** method should be used only if you want the application to be restarted.

The **HttpRuntime** object initializes a number of internal objects called helper objects. These objects include the cache manager and the file system monitor to detect application file changes. These objects help carry out page request tasks. When the **ProcessRequest** method is called, the **HttpRuntime** object starts working to serve a page to the browser. It creates a new empty context for the request and initializes a specialized text writer object in which the markup code will be accumulated. A context is given by an instance of the **HttpContext** class, which encapsulates all HTTP-specific information about the request. After that, the **HttpRuntime** object uses the context information to either locate or create a web application object that is capable of handling the request. The web application is searched using the virtual directory information contained in the URL. The object used to find or create a new web application is **HttpApplicationFactory**—an internal object responsible for returning a valid object that can handle requests.

USING THE **HTTPSERVERUTILITY** CLASS

Objects such as **Server**, **Request**, and **Response** are also included in the container represented by the **HttpContext** object. The feature-rich elements of these objects have been used extensively by ASP.NET developers.

We discuss these objects in detail, starting with the **Server** object. The functionality of the ASP-intrinsic **Server** object in ASP.NET is implemented by the **HttpServerUtility** class. An instance of the type is created when ASP.NET begins to process the request, and it is then stored as part of the request context. The set of helper methods that **HttpServerUtility** provides are publicly exposed to modules and handlers through the **Server** property of the **HttpContext** object. In addition, to maintain ASP.NET coding as close as possible to the ASP programming style, several other commonly used ASP.NET objects also expose their own **Server** property. You can use the object properties in their code in this way without encountering compile errors.

The **HttpServerUtility** class provides two properties: **MachineName** and **ScriptTimeout**. The **MachineName** property returns the machine name, and the **ScriptTimeout** property gets and sets the time (in seconds) that a request is allowed to be processed. This property accepts integers, and its default value is 90 seconds. However, if the page runs with the attribute **debug=true**, the value of this property is set to a very large integer so that a script can run before it is terminated, as shown in this code example:

```
this.Server.ScriptTimeout = 30000000;
```

The **ScriptTimeout** property is explicitly and automatically set in the constructor of the dynamically created class that represents the page.

Applying Skins and Themes to a Web Site

Themes are a collection of properties that you can apply on web pages and controls to maintain consistent styles across the pages in an application. The skin files can be part of the themes.

USING THEMES

Themes are made up of elements like cascading style sheets (CSS), images, and skins. There can be more than these elements in defining a theme. CSS users may find themes a little difficult to understand. However, themes are far superior to CSS in the sense that they can be applied across the entire application or at server level on the entire set of web applications. ASP.NET provides a number of readymade themes that are located at C:\WINDOWS\Microsoft.NET\Framework\vxxxxx\ASP.NETClientFiles\Themes. These themes are specified as strings that define the style and graphics as well as the CSS files for the application.

USING SKINS

Skins specify the property settings for all the standard controls within a web page, such as calendar, text boxes, buttons, and labels. Skins help define master pages as well as themes. Skin files have a file extension .skin and are located in the named Theme folders of an application. There can be multiple skin files for each control, or a single skin can define the theme for all the controls. The following code example shows the skin for a standard button control:

```
<asp:button runat="server" BackColor="lightblue"
ForeColor="black" />
```

This line of code can be saved as a .skin file in the Theme folder.

There are two types of skins:

- **Default skin:** A skin created specifically for a control without the `SkinID` attribute is called a default skin. When a theme is applied to a page, this skin is automatically applied on the controls of the specified type. Let's take the example of an already-created button skin. Because this is created as a default skin, it is applied to all the button controls present on a page when the corresponding theme is applied. Default skins are specific to controls and are not applied on controls for which they have not been defined.
- **Named skin:** A skin created specifically for a control with a `SkinID` is called a named skin. Named skins are not automatically applied on the control even after applying the theme on the page. A named skin needs to be applied explicitly by defining the control's `SkinID` property. You create named skins to define different styles for the same type of control on a web page.

USING CSS

Apart from images and skins, a theme can also include a CSS. When you put a .css file in the Theme folder, the style sheet is applied automatically when the theme is applied. Style sheets are defined in files with a .css extension and are saved in the Theme folder.

USING THEME GRAPHICS AND OTHER RESOURCES

Themes can also include graphics and other resources, such as script files and sound files. For example, part of your page theme might include a skin for a TreeView control, whose expand and collapse buttons can be specified as graphics in the Theme folder.

The scope of the themes encompasses two levels:

- **Web application level:** Themes defined at the web application level can be applied on each page by using the theme or the `StyleSheetTheme` attribute of the `@Page` directive. To avoid this redundant task of applying the theme on each page, you can also set the `<page>` element in the application configuration file. Remember, if the `<page>` element of the machine configuration file is set, the theme will be applied on all the web applications present in the server.

The page themes for a web application are saved in the respective subfolders under the \App_Theme folder of the application. The code example given next shows two themes, BluetHEME and PinkTHEME, in their folder structure in Solution Explorer:

```
MyWeb site
  App_Themes
    BlueTheme
      Controls.skin
      BlueTheme.css
    PinkTheme
      Controls.skin
      PinkTheme.css
```

CERTIFICATION READY?

Customize the layout
and appearance of a web
page.

7.1

- **Global level.** Global themes are used to define the overall style, look, and feel of all the web applications present on a server. Global themes are similar to page themes; the only difference is the location of the folder where they are stored. The theme files are accessible to all Web sites on the server and to all pages of the Web site.

Understanding ASP.NET Page Structure

An ASP.NET web page, with the file extension .aspx, contains three parts:

- **Directives:** The @Page directive sets up the environment for the ASP.NET parser and compiler to understand how the page should be processed and rendered. The page directive can have its property set to import namespaces or load assemblies or both.
- **Code:** This section contains the code to handle events of the page and controls. The code section is defined by the <script> tags. By default, the section defined by this tag is executed at the client side. If you need to execute the code at server side for your application, then you need to set the runat attribute of the <script> tag to server. Depending on the project type, .aspx page may also have a .design file, but can be ignored if not required.
- **Layout:** The page layout is defined within the <html> tag. Therefore, it is an HTML page, which includes the HTML body and its markup. The <body> tag can contain the client and server controls as well as simple text.

The ASP.NET pages have design files with .aspx and code-behind files with .cs in the file names. The code of the code-behind files are compiled before execution. In addition, all ASP.NET pages can be precompiled to an assembly if the assembly is the only file that needs to be deployed.

TAKE NOTE*

The following code example shows the code for a simple ASP.NET web page:

```
<!?page directives-->
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!--script-->
<script runat="server">

protected void btnMessage_Click(object sender, EventArgs e)
{
    Response.Write("Welcome " + txtInput.Text);
}

</script>

<!--layout-->
<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">
    <title>Welcome Page</title>
</head>
```

```

<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="txtInput" runat="server">
      </asp:TextBox>
      <asp:Button ID="btnMessage" runat="server"
      Text="Button"
      onclick="btnMessage_Click" />
    </div>
  </form>
</body>
</html>

```

In this example, the `runat` attribute is set to "server." This indicates that the code enclosed within the script block will be executed at the server. ASP.NET will create server-side objects to match the controls defined in the form and its attributes enclosed by this HTML tag. This server-side object is capable of running server-side code and raising server-side events.

To see the output, execute the page as you have seen here in the designing web pages with master pages section. Set this page as the start page. You can do this by right clicking on the page in Solution Explorer and selecting Set As Start Page. To execute the application, from the Debug menu select Start Debugging. The web page is displayed on the browser.

Apart from the `@Page` directive, a web page can use other directives, such as `@Assembly`, `@Implements`, `@Control`, `@Register`, `@PreviousPageType`, and `@MasterType`.

TAKE NOTE *

You can include the directives anywhere in the .aspx page or .ascx user control. However, as a best practice, you should include the directives at the beginning of the page.

CERTIFICATION READY?

Implement business objects and utility classes.

7.4

WORKING WITH IN-LINE CODING VERSUS CODE-BEHIND PROGRAMMING

The simple web page that was provided earlier contains all the code and markup in a single file. This is called in-line programming. Although this model is simple, and might be a logical model to choose when you are converting an ASP application to ASP.NET, it is recommended that you implement all new applications with the code-behind programming model. This is because the code-behind programming model provides a clean separation between the client-side and the server-side code. In addition, the code-behind programming model adds another file to the web page called the code-behind page. This page contains the server-side code, thus separating it from the client-side code and markup.

■ Understanding File Configuration and Compilation



THE BOTTOM LINE

The compilation model of web application project resembles the compilation model of the older version of Visual Studio 2008. All the project files, such as the code, designer class, and standalone files, are precompiled into a single assembly. The assembly file is built as a .dll file and persists in the bin folder. The assembly has the attribute such as assembly name, version, and location to specify the output location of the assembly.

The behavior of the ASP.NET application can be controlled by setting parameters at various levels.

Configuring Web Applications

System level settings are defined in the machine.config file. The application setting can be done in the web.config file of the application, which is stored in the root folder of the application.

USING .NET CONFIGURATION

All configuration files in ASP.NET are in XML format. The .NET runtime parses all the necessary configuration files, stores them in memory, and uses its parameters at runtime.

USING MACHINE CONFIGURATION

All the settings pertaining to the machine configuration of an application should be performed in the machine.config file. By default, this file is located at:

C:\Windows\Microsoft.NET\Framework\vxxxxx\CONFIG\machine.config.

(vxxxxx stands for the version of the framework installed on your system. If you have .NET Framework 3.5, it should be in the CONFIG folder of the v2 directory.)

It will be helpful if you know how information in the machine.config file is arranged. From the time of .NET 1.x, machine configuration has undergone substantial changes. In .NET 1.x, all the machine configuration information was bundled in a single machine.config file. However, in the later versions, the importance of segregating the information was realized. The browser definition capability files have been moved to a separate configuration file. In addition, comments are separated into a file named machine.config.comments.

It is important to maintain the comments file because it acts as a lookup resource that explains what configuration settings/changes the application has undergone.

As already discussed, configuration files are in the form of XML files, whose elements are divided into two types—the configuration section handler declaration and the configuration setting.

USING CONFIGURATION SECTION HANDLERS

The machine.config file stores the setting for the entire machine, and the ASP.NET application uses the web.config file to manage the setting at the application level. The machine.config file starts with the **configuration** tag that contains **configSections**, which holds the handlers for configuring .NET. These handlers are called configuration section handlers.

The machine.config file is located in the

C:\WINDOWS\Microsoft.NET\Framework\<.NET Framework version>\CONFIG folder.

Let's see how the machine.config file looks:

```
<configuration>
  <configSections>
    <section name="appSettings"
      type="System.Configuration.AppSettingsSection, System.Configuration,
      Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      restartOnExternalChanges="false" requirePermission="false"/>
    <section name="connectionStrings"
      type="System.Configuration.ConnectionStringsSection, System.Configuration,
      Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      requirePermission="false"/>
    <section name="mscorlib"
      type="System.Configuration.IgnoreSection, System.Configuration,
      Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      allowLocation="false"/>
  </configSections>
</configuration>
```

The authentication and authorization settings are also stored in similar fashion. A few handlers are displayed for you here. You can browse through the machine.config file located on your machine for the entire list:

```

<section name="authentication"
type="System.Web.Configuration.AuthenticationSection, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
allowDefinition="MachineToApplication"/>

<section name="authorization"
type="System.Web.Configuration.AuthorizationSection, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"/>

```

In this example, the authentication configuration settings are interpreted by the assembly with the string name:

`System.Web.Configuration.AuthenticationSection, System.Web,`
`Version=2.0.0.0, Culture=neutral,`
`PublicKeyToken=b03f5f7f11d50a3a` and the version and PublicKeyToken.

The strong name maintains the version of the assembly, along with the public token, which ensures that the assembly is not corrupted by any unwanted source. The ASP.NET framework has not changed many of its assemblies since .NET 2.0, as you can easily make out by looking at the version. It still refers to the older version 2.0.0.0.

USING WEB CONFIGURATION

You can control the behavior of your application by setting the parameters of the web.config file, which is similar to the how machine.config defines the behavior at the machine level. Suppose you want to set the sessionState exclusively for your application without affecting other applications on the server. You can do this by setting the sessionState in the web.config file. You can also use the `configSource` property in the web.config file to outsource specific sections of your configuration. The settings in the web.config file are only applicable at the application level, and they override the settings of the machine.config file.

The following code example shows the contents of a configuration file that turns on Forms authentication and tracing:

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.Web>
    <authentication mode="Forms" />
    <trace enable=true/>
  </system.Web>
</configuration>

```

Your application's configuration settings have actually been inherited from other web.config files. The machine.config file sets up the default .NET configuration settings. The top-level web.config file (in the .NET configuration directory) sets up the initial ASP.NET configuration. Then, the subsequent child web.config files within the request path can slightly modify the settings for individual applications.

Depending on the situation, you can tweak a few items and place the modified web.config file in your virtual directory and/or subdirectory. However, if different parts of your application need separate configurations, creating multiple web.config files is not a great idea.

CERTIFICATION READY?

Compile an application by using Visual Studio or command-line tools.

1.7

TAKE NOTE *

The ASP.NET configuration schema includes a location element to specify different settings for different directories, which can all be included in the master configuration file of your application.

Understanding the ASP.NET Compilation Model

Compiling an application offers many advantages. It improves the performance of the overall application. The execution of the compiled code is also much faster than the script languages ECMAScript because the code is compiled into assemblies and is ready to use.

Assemblies are mainly DLLs with compiled code that is closer to machine code, which means it does not require parsing. Compiled code is secure because it is difficult to reengineer; in other words, it is in low-level language and has low readability. Compilation also makes the source code stable because it has already been checked for syntax errors and type safety issues and is free of compile-time errors.

UNDERSTANDING THE COMPILED SEQUENCE

You need to compile your ASP.NET application before deploying it. To save time, you can also precompile the application before deploying. It is important for you to understand compilation and the order in which files and folders in the application are compiled. ASP.NET compiles an application in a specific order once a request is raised. The first set of compiled items is referred to as the top level. After the top-level items have been compiled, the folders, pages, and other items are compiled.

Top-level items are compiled in the following order:

1. App_GlobalResources
2. App_WebResources
3. Profile properties stored in the web.config file
4. App_Code
5. Global.asax

Once the top-level items are compiled, the next step compiles the ASP.NET folders, pages, and other items in the following order:

1. App_LocalResources
2. Web pages (.aspx files), user controls (.ascx files), HTTP handlers (.ashx files), and HTTP modules (.asmx files)
3. Themes, master pages, and other source files

When you compile an application for the first time, the assembly is also compiled for the first time, so it may be more time consuming. The next time you compile the already-compiled assembly will be used, which reduces your overall compilation time.

Additionally, an assembly is recompiled when you restart the application. This happens only when you modify the source code or add/remove/modify any top-level item or folder.

You can also restart an ASP.NET web application programmatically by unloading the application and then requesting a web page from that application after it has unloaded. This will then reload the application to give you the page you requested.

To restart an application programmatically, use the following code:

```
using System.Net;  
private void Restart()  
{  
    System.Web.HttpRuntime.UnloadAppDomain();  
    WebClient objWebClient = new WebClient();  
    string url = "http://localhost/Chapter1/Default.aspx";  
    byte[] stuff = objWebClient.DownloadData(url);  
}
```

In this code, a function `Restart()` is created, which can be called from any event, such as the `buttonClick` event.

MORE INFORMATION

To learn more about the compilation sequence and application restarts, refer to "ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0" on MSDN.

TAKE NOTE *

ASP.NET handles restart requests in a very efficient manner. It first serves all pending requests of the existing application domain along with the existing assemblies before restarting the application domain and loading new assemblies.

UNDERSTANDING DYNAMIC WEB SITE COMPIRATION

Dynamic compilation is the default compilation mode in ASP.NET. If you use dynamic compilation for your application, the Web site will not have any executable assembly that can be deployed on the server. The pages of the Web site will be compiled when requested. Alternatively, the precompiled page is dependent on the source file's timestamp. The page will be recompiled if a request is made after the page was changed.

The advantage of dynamic compiling is that it gives you the freedom to make changes to the codebase without forcing you to compile the code. If there are any modifications, ASP.NET automatically recompiles the source file and updates the linked resources. It is not necessary to restart the IIS server unless the `<processModel>` section has been changed. You can also use just-in-time compilation or JIT to compile your application. JIT is a method of dynamic compilation that helps improve the performance of an application. JIT compilation converts code at runtime into native machine code and caches the results of the translated code. This helps improve the overall performance of the application.

Dynamic compilation has certain disadvantages:

- It does not provide a compiled version of the site that may be required for deployment on a production server.
- It does not allow identification of compile-time errors, unless you access the Web site.
- The first request may get a slower response because the pages are compiled only on the first request.

COMPILING WEB APPLICATIONS USING THE COMMAND-LINE TOOL

You can compile an ASP.NET application before deploying it. This is known as precompilation. A web application created in ASP.NET can be precompiled using the `aspnet_compiler.exe` command-line tool. This tool is available only in ASP.NET 2.0 and its later versions. In addition, you can also use versioning to deploy the application on the server. It makes no difference whether the code is compiled using the command tool or the development environment. The final output of precompilation is an assembly created out of different source files, such as pages, user controls, resources, utility classes, and so on. To use the command-line tool for precompilation, you need to use the following command options:

- **-v:** Used if the application is to be deployed on an IIS server.
- **-p:** Used to specify the physical path of the files to be compiled.

If you use the `-p` option, `aspnet_compiler` treats all the subfolders within the application root folder as part of the root site. This may not be helpful if your web application is a collection of many subsites. In such cases, it is advisable to use the `-v` option.

CERTIFICATION READY?

Compile an application by using Visual Studio or command-line tools.

1.7

EXPLORE TELNET

Using Telnet, you can connect to a site and obtain the response from the site. You can install Telnet by using the Add/Remove Windows Components option on Windows Operating System. If you do not have Telnet installed, you can download Telnet from a number of Web sites, including <http://telnet-ftp-server.en.softonic.com/download>:

1. Start the command prompt from Start > All Programs > Accessories > Command Prompt. Alternately, in Start > Run print:
`Cmd`
2. Type the following command to clear the screen:
`Cls`
3. To start Telnet, type the following command at the command prompt and press the Enter key:
`telnet.exe`

This will display the following information on your window:

```
Welcome to Microsoft Telnet Client
Escape Character is 'CTRL1'
Microsoft Telnet>
```

- Help for this command can be obtained by typing the following command and pressing the Enter key:

```
?/
```

This will return the list of commands with explanation, as shown in Figure 1-6.

Figure 1-6

Telnet commands

```
Microsoft Telnet Client
Escape Character is 'CTRL1'
Microsoft Telnet> ?/
Commands may be abbreviated. Supported commands are:
c      - close          close current connection
d      - display        display operating parameters
o      - open hostname [port] connect to hostname <default port 23>
q      - quit           exit telnet
set    - set            set options (type 'set ?' for a list)
sen   - send           send strings to server
st    - status          print status information
unse - unset          unset options (type 'unset ?' for a list)
?h    - help           print help information
Microsoft Telnet> _
```

- To configure the setting, type the following command at the command prompt and press the Enter key:

Set bsasdel

The screen will display:

Telnet will respond as:

Backspace will be sent as delete

- You can revert by typing the following command:

Unset bsasdel

Telnet will respond as:

Backspace will be sent as backspace

- Now, open a connection with a Web site. Type the following command:

O myhost.com 80

Telnet will respond as:

Connecting to myhost.com.....



BUILD THE WEB APPLICATION AND ADD A WEB PAGE

- First create your own directory where you want to store your Web site. Let the folder name be c:\MyFirstSite. Next create a virtual directory in the IIS and map it to the site folder.
 - To open IIS, select Start > Settings > Control Panel. Then select Administrative Tools and start Internet Information Services (IIS) Manager. The node expands to form a tree. In the listed tree, on the left-hand side, you can see the "Default Web Site" node. The virtual directory is created under this node.
 - Now, right click the Default Web Site node. Select Add Virtual Directory from the context menu. IIS requests the name by which you want your site to be known. Name it MyFirstSite. Assuming that the name of the domain is CreateHostSite, the end user will need to type the following URL in the browser window:
<http://www.CreateHostSite.com/MyFirstSite>
- Note that CreateHostSite.com is a fictitious site. You can test this by accessing the site locally using the keyword **localhost**.

4. Map the virtual folder to the physical directory. The wizard will ask you for the physical drive path. You can browse to the folder c:\MyFirstSite and click ok to create the virtual site.
5. Create a simple HTML page. To do this, navigate to the folder c:\MyFirstSite and right click the folder. Select New > Text file. This adds a new text file to your folder. Rename it Welcomepage.html.
6. Type the following HTML code in the Welcomepage.html:


```
<!DOCTYPE html PUBLIC "...>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Untitled Page</title>
  </head>
  <body>
    <h1> Welcome to My Site</h1>
  </body>
</html>
```

Note: You can open the.html file in Notepad and type the HTML code.
7. To see how the page will appear in the browser, navigate to IIS as mentioned in Step 2, then select the Content View tab at the bottom of the main pane. When you locate the Welcomepage.htm files in the directory, right click the file and select browse. Alternatively, type the following URL in the navigation bar of the browser:
<http://localhost/MyFirstSite/Welcomepage.htm>
 The browser will display the page shown in Figure 1-7.

Figure 1-7

The result page



8. Convert the HTML file into an ASP.NET file to compare the two and know the difference. First, add the following page directive at the top of the file:


```
<%@ Page Language="C#" %>
```
9. Select File > Save As to change the extension of the file from .html to .aspx and rename the file as Welcomepage.aspx:

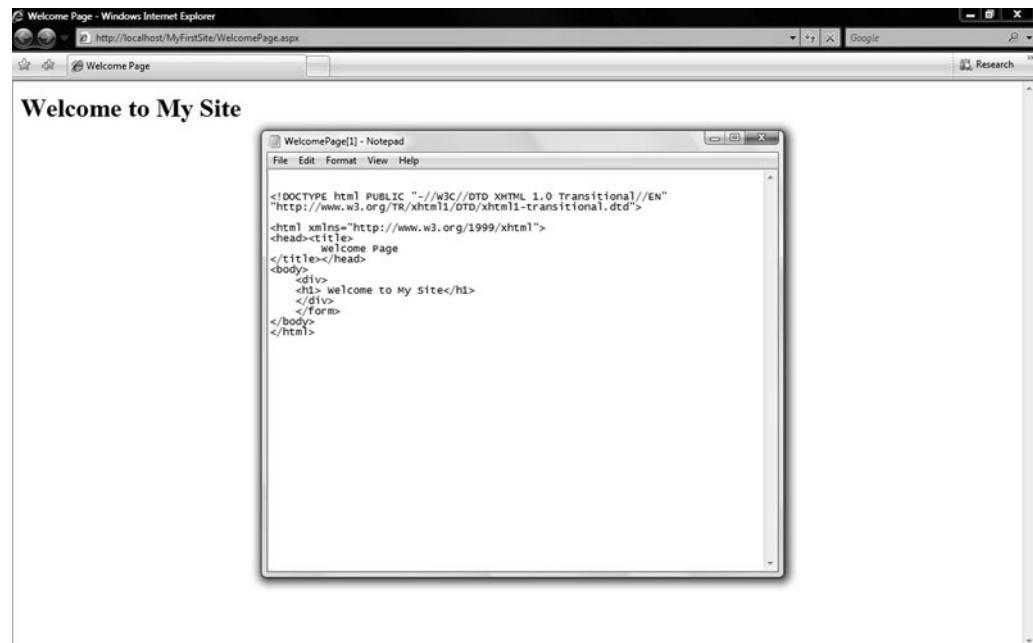

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "...>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
```

```
<title>Untitled Page</title>
</head>
<body>
<h1> Welcome to My Site </h1>
</body>
</html>
```

Note that both the pages look identical on the browser. If you right click on the page and select View Source, the HTML code will be displayed, as shown in Figure 1-8.

Figure 1-8

The result page



```
<!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>
<title>Welcome to My Site</title>
</head>
<body>
<h1> Hello World </h1>
</body>
</html>
```

Except for the page directive, the code is almost identical.

The page directive instructs the ASP.NET runtime to compile the code and base it on the Page class to interpret any code syntax encountered as C# code. In this case, you do not have any C# code on the page, but in later lessons, you will develop far more complicated pages with complicated logic using C#.

SKILL SUMMARY

ASP.NET 3.5 is shipped with .NET Framework 3.5 and Microsoft Visual Studio 2008. It is a technology for developing web-based applications. When compared to its previous versions, ASP.NET 3.5 has added features such as AJAX, new server controls, a new ListView data control, and extensions of server-based forms authentication, roles management, and profile services.

(continued)

SKILL SUMMARY (*continued*)

Web servers are servers where applications are deployed. They receive and handle requests from browsers via HTTP. The web browser acts as a medium to display and allow users to interact with web pages on a Web site. IIS works as a watchdog for applications on the Microsoft platform and intercepts HTTP requests. HTTP is a text-based protocol that defines how web browsers and web servers communicate. An HTTP request is the request sent by the browser to a server through a URL. Responses sent by the server are in form of HTTP response objects.

The life cycle of an ASP.NET application has many stages. It starts when the request is sent by the browser to the server and ends with the request close. It is important to understand the integral components of ASP.NET like projects, solutions, and assemblies. All items required by a Web site are grouped at two levels—solution and project. A solution container contains multiple projects, but a project container typically stores multiple items. Using these containers, you can manage the settings for your solution as a whole or for individual projects. Assemblies act as primary units for version control, deployment, and security in the .NET Framework.

ASP.NET 3.5 helps create Web sites with ease and enhanced features, such as the addition of master pages, nesting of master pages, and use of themes and skins. Master pages are similar to web pages, except that they have @Master directives. The first part of an ASP.NET page structure contains the page directives, the second part is the code, and the last part is the page layout. ASP.NET page coding is done with the help of ASP-intrinsic objects, such as `HttpApplication`, `HttpRequest`, `HttpRuntime`, `HttpContext`, and `HttpServerUtility`.

When a request is executed for the first time, ASP.NET compiles the application components in a specific order. The items compiled first are referred to as the top-level items. These top-level items are recompiled only when there is any dependency change. The primary advantage of compiling an application is that it makes execution faster. There are different types of compilation, such as dynamic Web site compilation and precompilation. You can precompile your Web site using the `aspnet_compiler` tool.

For the certification examination:

- Configure application pools.
- Handle events and control page flow.
- Configure projects, solutions, and reference assemblies.
- Work with ASP.NET intrinsic objects.
- Customize the layout and appearance of a Web page.
- Implement business objects and utility classes.
- Compile an application by using Visual Studio or command-line tools.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. An _____ can span multiple files, therefore, it is often called a logical DLL.
2. The _____ directive sets up the environment for the ASP.NET parser and compilers to understand how the page should be processed.
3. A skin created specifically for a control without the `SkinID` attribute is called a _____ skin.
4. Result or information from server to the web browser is called _____.
5. IIS allows configuring IIS extension dynamic-link libraries (DLLs) called _____.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---|--|
| T | F 1. Dynamic compilation provides a compiled version of the site that may be required for deployment on a production server. |
| T | F 2. A global.asax file is created at the root directory of the application and contains the code to handle application events. |
| T | F 3. GET and POST are commonly used HTTP commands. |
| T | F 4. The <code>HttpExtensionProc</code> method has the <code>EXTENSION_CONTROL_BLOCK</code> structure, which includes all the context of the request. |
| T | F 5. IIS intercepts HTTP requests from port 80. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. You are making enhancements to a web application developed in ASP.NET 3.5. When you try to compile your application, the compiler throws an error that the assembly reference is missing. What should you do to successfully compile the application?
 - a. Change the output path in the property pages of the project to the location of the missing assembly.
 - b. Add a reference of missing assembly in the property pages of the project.
 - c. Delete the assembly reference, and add a reference to the missing assembly by browsing for it on your hard disk.
 - d. Add a working directory in the property pages of the project to the location of the missing assembly.
2. You have finished working on an application, and you want to compile it. Which of the following statement is true about compiling an application for the first time?
 - a. Compiling an application takes place only once. From there on, only interpretation of the application's source code takes place.
 - b. Compiling an application for the first time takes the same time as the subsequent compilations because the assemblies are all recompiled every time the application is reloaded.
 - c. First time compilation of an application does not include compilation of assemblies.
 - d. When you compile an application for the first time, it may take a longer time to compile than when you compile at other instances. This is because during this compilation, the assembly is compiled too, which can be reused for subsequent compilations.
3. When configuring your application, you want to retrieve certain configurations from an external include file. To do this, which property should you use and where should this be set?
 - a. `configSource` property in the `web.config` file
 - b. `configOutSource` property in the `web.config` file
 - c. `sessionState` property in the `machine.config` file
 - d. `configSource` property in the `machine.config` file

Matching

Match the term in Column 1 to its description in Column 2.

- | | |
|------------|---|
| a. TRACE | _____ 1. Allows a client to directly create a resource at the indicated URL on a server. |
| b. POST | _____ 2. Allows the client to see what is being received at the other end of the request chain. |
| c. HEAD | |
| d. PUT | |
| e. CONNECT | |

- _____ 3. Creates a new, dynamically named resource. Data retrieved using this method is not cached.
- _____ 4. Reserved for use with a proxy that can dynamically switch to being a tunnel, such as the Secure Socket Layer (SSL) protocol.
- _____ 5. Retrieves the metadata of a resource, which is identical to the information sent in response to a GET request.

Review Questions

1. What is the importance of the `HttpApplication` class?
2. What are the three important aspects of an ASP.NET web page?
3. As compared to in-line coding, what are the advantages of a code-behind programming model?

■ Case Scenarios

Scenario 1-1: Understanding Project Folder Structure

You are creating an HR solution application for a software company using Microsoft ASP.NET 3.5. The application has to interact with SQL Server 2005. To do this, you have XML files, utility classes, and business objects. Your application also needs to support a GUI that has a uniform look and feel. To do this, you have master pages, themes, and skins.

Under which folder will you place these files and what is the significance of their location?

Scenario 1-2: Designing Page-Level and Application-Level Master Pages and Themes

You are working on a huge Web site of a product company. It has many subsites for vendors, end customers, and other entities. All of these should look similar to give the user a holistic feel. For example, all the button controls of the application should have the same skin.

How will you create the master pages and theme pages for the entire site and its subsites?



Workplace Ready

The Importance of Assemblies

The introduction of assemblies by Microsoft in the .NET Framework has brought a shift in the way executable files are handled, shared, and distributed. Assemblies are the most integral part because they are the building blocks of .NET Framework programming. In simple words, they are nothing but the executable files of .NET Framework such as DLL or EXE files.

You may wonder what makes assemblies the most predominant part of .NET Framework applications. If you are familiar with older languages like VB6 or C++, then you might easily recollect how the components were built, installed, and copied to a new location. If, at the time of installing, the registry maintained an older version of the same component, it led to a version conflict known as “DLL Hell.” The only way to resolve the conflict was to break into the system and remove the unwanted version from the registry. How do you think this is handled in .NET Framework assemblies?

Creating and Configuring Server Controls

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Working with web pages and controls.	Handle events and control page flow.	7.6
Using standard web server controls.	Consume standard controls.	2.5
Using specialized web server controls.	Consume standard controls.	2.5
Creating data-bound controls.	Bind controls to data by using data-binding syntax.	3.5
Creating collections.	Implement data-bound controls.	2.1

KEY TERMS

postback

request

response

server controls

view state

web page life cycle

■ Introducing Web Pages and Controls



THE BOTTOM LINE

Developing a web application involves creating several web pages that include various types of controls. To develop a full-fledged application, you should be aware of the different stages of the life cycle of a web page and the steps to use HTML **server controls** and web server controls.

Understanding the Life Cycle of a Web Page and Its Controls

The life cycle of a web page starts when the web browser requests the page from a Web site. The web server renders the page and its child controls. The rendered page is then sent to web browser. The web page and its child controls objects are then destroyed to free up the resources on the server.

UNDERSTANDING LIFE CYCLE STAGES AND EVENTS

Every server control has methods and events that are executed during the *web page life cycle*. Because web pages are derived from the `Control` class, these methods and events are executed when the controls are created and destroyed. The life cycle of an ASP.NET page starts when the user sends a request for a page. On receiving the *request*, ASP.NET evaluates whether it needs to run the page or can send the cached page back. If not, it then decides whether the page should be compiled or parsed.

The page life cycle begins with the start stage. In this stage, ASP.NET checks whether the request is a new request or a postback and sets the page properties accordingly.

Once the page properties are set, the controls are loaded, and the properties for the controls are set. This is known as the page initialization stage. After page initialization, the page goes through other stages, such as load, validation, postback event handling, and rendering. The page is finally unloaded during the unload stage, when the page, after being sent to the client, is cleaned and ready to be discarded. The server sends the *response* to the client after processing the request.

While a web page goes through various stages in its life cycle, it also raises different events. You can handle these events to run your own code.

The most frequently used events of the web page life cycle are `PreInit`, `PreLoad`, `Load`, `PreRender`, `Render`, and `Unload`.

MORE INFORMATION

Submitting data back to the server for further processing is called a *postback*. To learn more of the details about web page life cycle stages and the events raised at each stage, refer to the [ASP.NET Page Life Cycle Overview](#) section on MSDN.

WORKING WITH VIEW STATE

Events in the web page life cycle may cause data loss when the user wants to hold on to the data or state of the object between server calls. In such situations, *view state* helps.

View state of a control contains all the control's property values that need to be preserved across HTTP requests. View state is stored as a variable in an HTML hidden input element and accessed when page requests are processed.

Any object data that cannot be represented as HTML in the web page can be placed in the view state. For example, if a postback occurs in a web page's Submit button `click` event, then all form data gets sent back to the web server.

To store this data, view state is implemented with the help of hidden form fields called `_ViewState`. On the server side, the view state is used to re-create the web page and its server controls.

Loading the view state with excess data can cause performance issues because this data is sent back and forth between the browser and server. Set the `EnableViewState` property to `true` for those server controls values you want to keep. For the rest, set the property to `false` and use control states to retrieve the values.



For more information on control states, refer to Lesson 7, "Programming Web Applications."

CERTIFICATION READY?

Handle events and control page flow.

7.6

Differentiating between Server and HTML Controls

Developing a web application with seamless communication between the web browser and web server is a challenging task for a web application developer. "Seamless" means the ability to bring data back and forth in such a way that the user gets the impression that the server and the browser are the same.

To create such an application, Microsoft ASP.NET provides server controls that establish automatic communication with the web server when an event occurs. There are two types of server controls: HTML and web.

HTML server controls are essentially HTML elements that are processed by the server. An HTML server control is processed before sending a web page to the user or when the page is posted back to the server.

HTML server controls are mapped directly to the HTML elements, and their properties are identical to the corresponding HTML element's attribute.

The following code example shows you how to add an HTML server control in a web page:

```
<html>
    <head>
        <title>Send Email</title>
    </head>
    <body>
        <form name="InputForm" method="post" id="frmInputForm"
runat="server">
            <input type="text" name="EmailAddress" id="txtEmail"
runat="server" >
            <input type="submit" name="PostIt" value="Send Email"
id="submitEmail" runat="server">
        </form>
    </body>
</html>
```

HTML server controls have the **runat** attribute. The value of this attribute is set to **server**, which means that these controls are created by modifying the elements of the HTML documents.

HTML server controls are useful when controls need to have client-side JavaScript events attached. Each HTML server control is mapped to a single HTML tag. The HTML element cannot be referenced in server-side code. You need to convert them to HTML server controls. To use these controls in the code behind, make use of the **Id** attribute.

Some of the widely used properties of HTML server controls are:

- **Disabled:** A Boolean value that enables or disables a control. Its default value is set to **false**.
- **Id:** Provides a unique id for the element.
- **Name:** Indicates the name of the element.
- **Runat:** Specifies where the element is processed. Server controls have the value for this property set to “server.”
- **Style:** Sets or gets the CSS properties applied to this control.
- **Tagname:** Returns the element tag name.
- **Type:** Indicates the type of the element.
- **Value:** Indicates the value of the element.
- **Visible:** A Boolean value that hides or makes the element visible in the web page. Its default value is set to **true**.

Web server controls provide a more consistent programming model compared to HTML server controls. HTML server controls follow an HTML-centric model, not an object-oriented model, which is more consistent. In addition, HTML server controls can't render themselves based on the browser capabilities; therefore, you need to write code for browser compatibility. In contrast, ASP.NET web server controls can render themselves based on the browser capability. HTML server controls require scripting language support for validation, whereas ASP.NET server controls have a built-in feature for validation and also support an

object-oriented programming model. Web server controls can generate as many HTML tags as needed and can include client-side JavaScript. They can detect browser capability and render content appropriately. In addition, they provide a rich object model with type-safe programming capabilities.

Web server controls fall under a hierarchy of classes and are derived from the `WebControl` class. Like all classes, the look and behavior of a web server control is also controlled by its properties. Properties and attributes are mapped in the markup of the web server control in the .aspx file and may not render as attributes in HTML that is rendered to the web browser. Attributes provide additional information about a control while methods are subroutines that perform some actions. Methods can be associated with a class or object, but properties show the state of an object.

The most widely used properties of web server controls are:

- **`Id`**: Indicates the programmatic identifier assigned to the control.
- **`Height`**: Indicates the height of the web server control.
- **`Width`**: Indicates the width of the web server control.
- **`TabIndex`**: Indicates the order in which controls in a page can be accessed using the tab key.
- **`Visible`**: A Boolean value that indicates whether a web server control is rendered on the web page.
- **`Enabled`**: A Boolean value that indicates whether a web server control is enabled on the web page.
- **`Font`**: Specifies the associated font properties for the web control.

Web server controls can be added to a web page dynamically. The following code example in C# shows you how to add a button on initialization of a web page:

```
protected void Page_Init(object sender, System.EventArgs e)
{
    Button MyButton = new Button();
    MyButton.ID = "btnClickMe";
    MyButton.Text = "Click Me";
    MyButton.Click += newEventHandler(MyButton_Click);
    this.Form1.Controls.Add(MyButton);
}
void MyButton_Click(object sender, EventArgs e)
{
    // TODO: Add your Logic here.
}
```

X REF

For more information on control states, refer to Lesson 1, “Introducing ASP.NET 3.5.”

TAKE NOTE*

If you want to distinguish between local and member variables you should use “this.”

This code example adds a new button `btnClickMe` to `Form1` in the `OnInit` method. You can add your own code to execute a set of steps in the `click` event handler of this button. `OnInit` is the method that raises the `Init` event, whereas `Page_Init` is the event handler attached to the `Init` event.

HTML controls provide a simple object model to handle properties and attributes of HTML elements at server side. This is useful during migration from classic ASP to ASP.NET. In addition, to manipulate the properties and attributes at the server side, the logic that affects the attributes of the control can be moved to server-side event handlers.

+ MORE INFORMATION

For information on creating a web form and handling events, refer to the How to: Add ASP.NET Web Pages to a Web Site section on MSDN.

■ Using Standard Web Server Controls



While creating web pages, you often use various basic web server controls, such as Label, TextBox, and Button control to take input from users or display data. These controls are easy to use and provide various properties and events that you can modify or set to customize the controls.

Using the Label Control

You can use the Label control to display text or change text dynamically on a web page when the control is rendered.

For instance, you can use a Label control to display the number of hits on a web page every time a user logs on. You can also use a Label control to provide captions or text for other controls on the web page. This control gets rendered to a `` tag.

You can assign a string value or HTML content to the `Text` property of the Label control. Alternatively, including text placed between opening and closing tags of the Label control also gets assigned to the `Text` property of the Label control.

The Label control supports Cascading Style Sheets (CSS). It may be a good idea to make use of a CSS to format the rendered text instead of using the properties of the control.

The Label control is derived from the `WebControl` class and the `ITextControl` interface. This interface contains the definition that the control implements to get or set its text content.

Some important properties of the Label control are:

- **Text:** Denotes the text to be shown on the label.
- **EnableViewState:** Decides if the control automatically saves its state for use in round-trips. The default value of this property is `true`.
- **ID:** Indicates the programmatic name of the control.

Using the TextBox Control

You use the TextBox control to get inputs from the user. The `Text` property of this control is used to get or set the content of the control. The TextBox control gets rendered to `<input type="text"/>` tag.

The TextBox control is derived from the `WebControl` class and three interfaces: `IPostBackDataHandler`, `IEditableTextControl`, and `ITextControl`. The `IPostBackDataHandler` interface defines the methods that ASP.NET server controls must implement to automatically load postback data. The `IEditableTextControl` interface represents a control that renders text that can be changed by the end user. The `ITextControl` is an interface that a control implements to get or set its text content.

Some frequently used properties of the TextBox control are:

- **TextMode:** Represents the behavior mode of the text box. Its value can be `SingleLine`, `MultiLine`, or `Password`. Single line allows one line of text entry, and multiline allows a full paragraph of text to be entered. Password text boxes allow a single line of text entry with masks. The default value of this property is `SingleLine`.
- **Columns:** Represents the width of the text box in characters.
- **MaxLength:** Denotes the maximum number of characters that can be entered in the text box.

- **ReadOnly:** Specifies whether the text in the control can be changed. This property accepts the Boolean values `true` or `false` and allows users to set the text box focus. They can copy the text but can't modify it. However, the `Enabled` property when set to `false`, neither allows users to set the focus nor copy the text.
- **Wrap:** Indicates whether the text entered can be wrapped to the next line automatically. It accepts the Boolean values `true` or `false`.
- **AutoPostBack:** Performs automatic postback of the data when the text is modified. It accepts the Boolean values `true` or `false`; the default is `false`.
- **Enabled:** Indicates the enabled state of the control.
- **EnableViewState:** Specifies whether the control automatically saves its state for use in postback.
- **CausesValidation:** Indicates whether the validation method is fired on text entry or modification. It accepts Boolean values with the default value as `false`.

Using the Button Control

Button controls are push buttons that appear on web pages. This control is derived from the `WebControl` class, and it implements the interfaces `IButtonControl` and `IPostBackEventHandler`. The `IButtonControl` interface defines properties and events that must be implemented to allow a control to act as a button on a web page. The Button control gets rendered as `<input type="submit">` tag.

A Button control can typically be a Submit or Command button. The `Click` event of a Submit button initiates a postback to the server and does not need the `CommandName` property to be set. You can use the `Click` event handler to control the actions that should be performed as an outcome of the event.

For a Command button however, you need to set the `CommandName` property. Setting this property enables you to create multiple Command buttons and also to trap which button is clicked in the event handler. You can also use the `CommandArgument` property to provide additional information about the command. For example, to sort data, you can set the `CommandName` to `Sort`. In addition, you can use the `CommandArgument` property to decide whether you want to sort in descending or ascending order. You can also use Command buttons to play or pause animation or music files.

Some commonly used properties of the Button control are:

- **CausesValidation:** Holds a Boolean value that determines whether the button causes any validation to be fired. The default value of this property is `true`.
- **Enabled:** Indicates the enabled state of the control.
- **EnableViewState:** Contains a Boolean value that determines whether the control automatically saves its state for use in round-trips to the server.
- **OnClientClick:** Indicates the client-side script that is executed on a client-side `Click` event. This occurs before the page is posted back.
- **PostBackUrl:** Specifies the URL to which data should be posted when the button is clicked. This is an optional property; by default, the page is posted back to the current page.
- **UseSubmitBehavior:** Holds a Boolean value indicating whether the button renders as a Submit button. The default value of this property is `true`.

TAKE NOTE*

The `UseSubmitBehavior` property gets or sets a value that represents whether the Button control uses the submit mechanism of the client browser or the ASP.NET postback mechanism.

Using the CheckBox Control

A CheckBox control allows users to select a `true` or `false` condition value. This control is derived from the `WebControl` class and implements the interfaces `ICheckBoxControl` and `IPostBackDataHandler`.

The **ICheckBoxControl** interface defines the property and event that a control implements to act as a check box. This control gets rendered as `<input type="checkbox">` tag.

Some important properties of the CheckBox control are:

- **Checked:** Holds a Boolean value indicating the checked state of the control. The default value of this property is `false`.
- **AutoPostBack:** Contains a Boolean value indicating whether data is posted back to the server automatically when the control is clicked.
- **CausesValidation:** Indicates whether the control causes validation to be fired.
- **EnableViewState:** Specifies whether the control automatically saves its state for use in round-trips to server.
- **Text:** Indicates the caption or label shown with the check box.
- **TextAlign:** Denotes the alignment of the text with respect to the check box item. The default alignment is right.

The **CheckedChange** event fires when the checked state of the control changes. You can add code for execution in this event method.

TAKE NOTE *

Typically, a **CheckBoxList** control renders text items as an HTML table. If you do not want the text items to appear as an HTML table, set the **RepeatLayout** property of the control to **Flow**.

List controls are easier to bind to data rather than to a group of individual controls. It is easier to bind data using **CheckBoxList** control rather than using a group of **CheckBox** controls.

Using the RadioButton Control

A **RadioButton** control is used to accept user input on a web page or form. This control enables the user to select only one option among the multiple options that may be provided. You can use the **GroupName** property to specify a name for the **RadioButton** controls for easy identification. The **RadioButton** control is rendered using the `<input type="radio">` tag.

The **RadioButton** control is derived from the **CheckBox** class and implements the **IPostBackDataHandler** interface.

Some important properties of the **RadioButton** control are:

- **AutoPostBack:** Holds a Boolean value indicating whether automatic posts back to the server can be performed when the control is clicked. The default value of this property is `false`.
- **CausesValidation:** Contains a Boolean value indicating whether the control causes validation to be fired.
- **Checked:** Indicates the checked state of the control. The default value is `false`.
- **EnableViewState:** Indicates whether the control automatically saves the control state for use in the server round-trips. The default value is `true`.
- **GroupName:** Represents the group to which the radio button belongs.
- **Text:** Indicates the caption text displayed with the radio button.
- **TextAlign:** Specifies the alignment of the text with respect to radio button item. The default alignment is right.

The **CheckedChange** event fires when the checked state of the control changes. As web application developers, you can add your code for execution in this event method. Figure 2-1 shows the various web server controls.

Figure 2-1

Web server controls

**TAKE NOTE ***

List controls are easier to bind to data rather than to a group of individual controls. It is easier to bind data using **RadioButtonList** control rather than using a group of **RadioButton** controls.

CREATE A SIMPLE WEB PAGE

WARNING Note that the controls in your web application need to be populated with data only from trusted sites. For example, populating a CheckBox and a RadioButton control with data from an untrusted source can cause an XSS attack. It is therefore, advisable to encode data from untrusted sources using the **Server.HtmlEncode** or **HttpUtility.HtmlEncode** property of the **HttpUtility** class.

Note that the **MaxDataBindDepth** property specifies the maximum depth to which the **TreeView** will bind the data. The **TreeView** control performs the client-side script callbacks while expanding nodes. Malicious users can craft a callback to get the **TreeView** data that you might have tried to hide using the **MaxDataBindDepth** property. Therefore, you should never use the **MaxDataBindDepth** property to hide sensitive data.

Now that you have learned about the basic server controls, let's create a web application that has a simple login page that asks users for their username and password. The page provides a Login button to log on and a Remember Me check box to remember the user details for subsequent logons.

To create this web page:

1. Add a new web form **Login.aspx** in your application.
2. From the Toolbox, drag and drop a Label and a TextBox control for entering the username. Set the properties as shown in this markup:

```
<asp:Label ID="lblUserName" runat="server" Text="User Name">
</asp:Label>
<asp:TextBox ID="txtUserName" runat="server"
height="22px" width="128px">
</asp:TextBox>
```

This will display the User Name label and the text box where users can type in their user name.

3. Add another Label and a TextBox control for entering the password. Once you set the properties, the following code is generated:

```
<asp:Label ID="lblPassword" runat="server" Text="Password">
</asp:Label>
```

```
<asp:TextBox ID="txtPassword" runat="server" TextMode="Password">
</asp:TextBox>
```

This will display the Password label and a text box that allows users to enter their password.

4. Add a CheckBox control on the web form. Once you set the properties, the following code is generated:

```
<asp:CheckBox ID="chkRemember" runat="server"
Text="Remember Me"/>
```

5. Add a Button control to the web form. Once you set the properties, the following code is generated:

```
<asp:Button ID="btnLogin" runat="server" Text="Login"
onclick="BtnLogin_Click" />
```

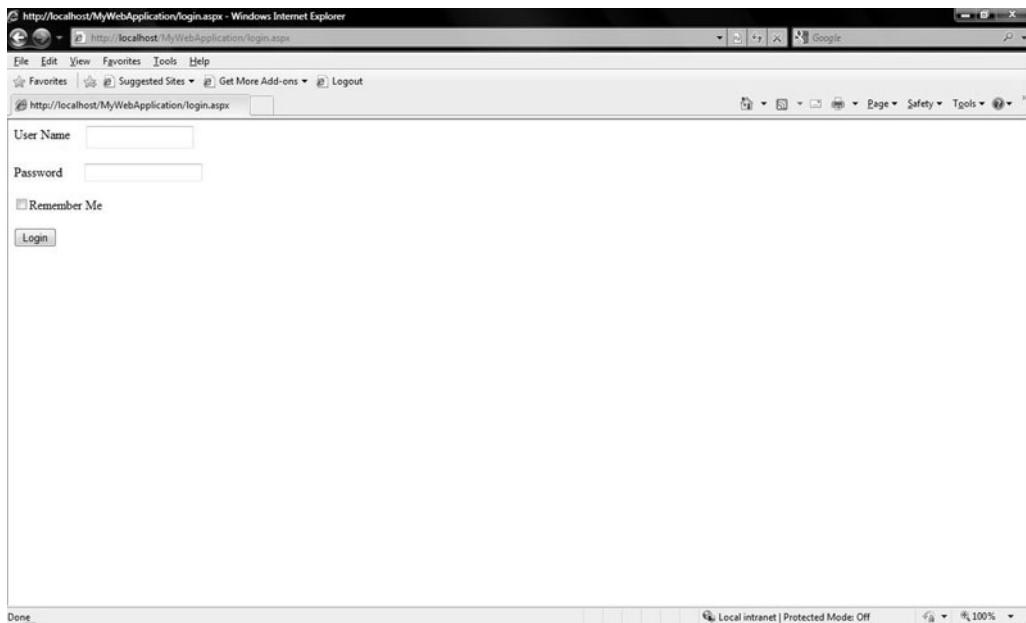
Here is the complete markup for the Login.aspx file:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Login.aspx.cs" Inherits="MyWebApplication.Login" %>
<!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></title>
</head>
<body>
<form id="form1" runat="server">
<asp:Label ID="lblUserName" runat="server" Text="User Name">
</asp:Label>
 &nbsp;&nbsp;&nbsp;
<asp:TextBox ID="txtUserName" runat="server"
height="22px" width="128px">
</asp:TextBox>
<br />
<br />
<asp:Label ID="lblPAssword" runat="server"
Text="Password">
</asp:Label>
 &nbsp;&nbsp;&nbsp;&nbsp;
<asp:TextBox ID="txtPassword" runat="server"
TextMode="Password">
</asp:TextBox>
<br />
<br />
<asp:CheckBox ID="chkRemember" runat="server"
Text="Remember Me"/>
<br />
<br />
<asp:Button ID="BtnLogin" runat="server" Text="Login"
onclick="BtnLogin_Click" />
</form>
</body>
</html>
```

Figure 2-2 shows the Login web page in the design view.

Figure 2-2

Login web page



6. To hold the input values, add a new class `UserCredentials` in your application, with the public properties `UserName`, `Password`, and `RememberUser` as shown:

```
public sealed class UserCredentials
{
    public string UserName
        { get; set; }
    public string Password
        { get; set; }
    public bool RememberUser
        { get; set; }
}
```

7. To get the values from the username and password text boxes and the Remember Me check box, and to populate the object of the `UserCredential` class and then pass it on for authentication, write the following code in the `BtnLogin_Click` method:

```
protected void BtnLogin_Click(object sender, EventArgs e)
{
    UserCredentials uc = new UserCredentials();
    uc.UserName = txtUserName.Text;
    uc.Password = txtPassword.Text;
    uc.RememberUser = chkRemember.Checked;
    // TODO: Pass the object of UserCredentials to the
    Authentication Layer.
}
```

Note that you can pass the user name and password as string variables to the authentication layer. Creating a business object like `UserCredentials` offers a better approach especially when you want to encrypt and decrypt the data or even serialize and deserialise it. As a good programming practice, you should never use the password in plain text format. You should always encrypt it using some mechanism before you use it in your application.

Note that in the previous example that you added the `Click` event handler syntactically in the `.aspx` file. Alternatively, you can add the `Click` event handler dynamically in the `Page Load` event as shown here:

CERTIFICATION READY?
Consume standard
controls.
2.5

```
protected void Page_Load(object sender, EventArgs e)
{
    this.BtnLogin.Click += new EventHandler(BtnLogin_Click);
}
```

You add the event handlers dynamically when you create the controls at runtime.

■ Using Specialized Web Server Controls



THE BOTTOM LINE

While developing web applications, you might want to display data to users in different formats, such as tables or lists. For instance, if a web application calculates a loan repayment schedule after receiving the loan amount and tenure as input from the user, it needs the data to be displayed in tabular format. A web application that provides users with possible baby names needs to display data in list format. A web application that provides online dating services to users needs to allow users to upload files and also provide them with a date picker functionality. To cater to such requirements, ASP.NET provides various web server controls such as Table, TableRow, TableCell, HotSpot, Calendar, Panel, Image, and FileUpload.

Displaying Data in Tables

TAKE NOTE *

The Table, TableRow, and TableCell controls are new in ASP.NET 2.0 and higher versions. Dynamically added rows and cells are not persistent during post-back. Therefore, Table controls are best suited for the development of custom controls.

ASP.NET provides specialized Table, TableRow, and TableCell web server controls to display the data in tables.

You can use the Table control to display data in a tabular format on a web page. A Table control can hold and display text, images, as well as additional controls. A Table control consists of a collection of rows known as TableRow controls. These TableRow controls in turn make up a collection of cells, known as TableCell controls.

Using a Table Control allows you to add more TableRow and TableCell controls to the table at runtime. However, during postback, these TableRows and TableCells are re-created.

The Table, TableRow, and TableCell controls are derived from the `WebControls` class and therefore inherit formatting properties such as `Font`, `Backcolor`, and `ForeColor`.

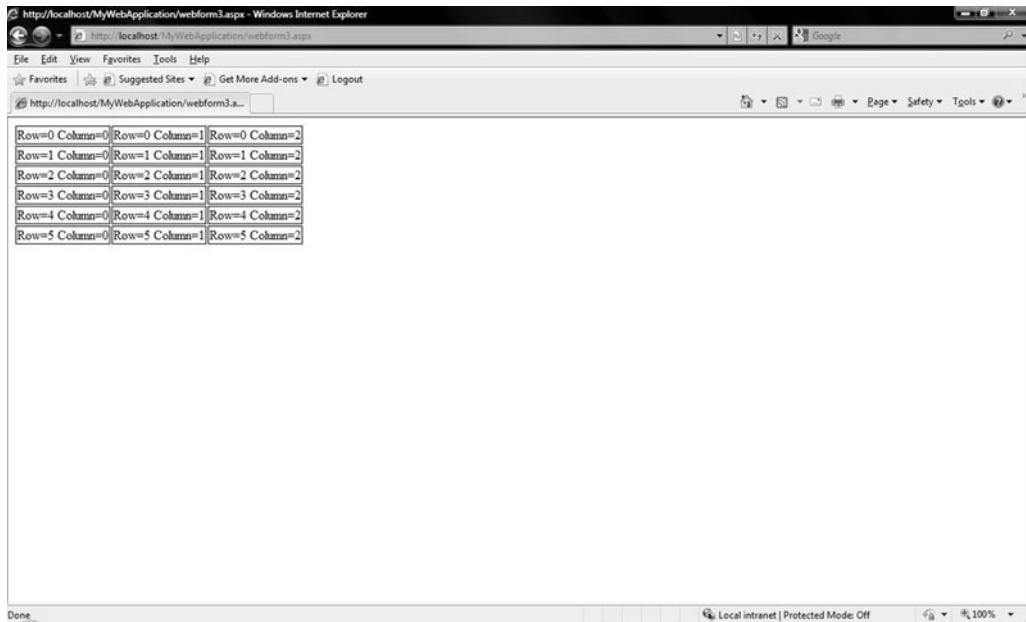
The following code snippet shows you how to add a Table, TableRow, and TableCell control dynamically:

```
protected void Page_Load(object sender, EventArgs e)
{
    Table myTable = new Table();
    TableRow row;
    TableCell cell;
    for (int i = 0; i < 6; i++)
    {
        row = new TableRow();
        for (int column = 0; column < 3; column++)
        {
            cell = new TableCell();
            cell.Text = "Row=" + i.ToString() +
                "Column=" + column.ToString();
            cell.BorderWidth = 1;
            row.Cells.Add(cell);
        }
        myTable.Rows.Add(row);
    }
    form1.Controls.Add(myTable);
}
```

This code creates a table of 6 rows and 3 columns. For every row, you need to add a `TableCell1` that indicates the number of columns, and then add this `TableRow` to the `myTable` object. Figure 2-3 shows the output of the program.

Figure 2-3

A table created using Table control



USING THE CALENDAR CONTROL

A Calendar control is included in a web page to receive a date that the user has selected through navigation. This control provides support for all types of calendars, such as Gregorian and Hijri, with the help of the `System.Globalization` namespace.

You can use the `SelectionMode` property to enable the user to select a day, week, or month. The selected date can be retrieved using the `SelectedDates` collection.

By default, the Calendar control displays the days of the month, the day headings for the days of the week, the titles for the months and the year, and the links for each day and for the next and previous months. However, you can customize these properties and styles by using the following properties:

- **DayHeaderStyle:** Specifies the style for the section that displays the days of the week.
- **DayStyle:** Specifies the style for the dates in the displayed month.
- **NextPrevStyle:** Specifies the style for the navigation controls in the title section.
- **OtherMonthDayStyle:** Specifies the style for the dates that are not in the currently displayed month.
- **SelectedDayStyle:** Specifies the style for the selected dates on the calendar.
- **SelectorStyle:** Specifies the style for the week and month date-selection column.
- **TitleStyle:** Specifies the style for the title section.
- **TodayDayStyle:** Specifies the style for the current date.
- **WeekendDayStyle:** Specifies the style for the weekend dates.

Calendar controls do not support binding of data to a data source. They only display the calendar and create and assemble the components that make up the control.

TAKE NOTE *

The Calendar control renders the ECMAScript to browser. Therefore, you should customize the browser to support ECMAScript/JavaScript. As a developer, you can't control the script setting from your code, so the functionality of such controls depends on how users configure their browser settings.

Displaying Images

ASP.NET provides the ImageButton and ImageMap controls with ImageButton and ImageMap subclasses to display images on a web page. This section will discuss using these controls.

USING THE IMAGEBUTTON AND IMAGEMAP CONTROLS

Say that you are creating an online album to display the photographs of new models for an advertising agency. You want your users to browse through the photographs and select the model they want to use for advertisements. The users should also be able to view a thumbnail of the photographs. To build such an application, you would use the Image, ImageMap, and ImageButton controls.

The `Image` class is derived from the `WebControl` class. The `Image` controls do not raise a `Click` event. The `Image` control renders the `` element on a web page.

Some important properties of an `Image` control are `ImageUrl`, `Alternate Text`, `DescriptionUrl`, `ImageAlign`, and `GenerateEmptyAlternateText`.

Both `ImageMap` and `ImageButton` controls are derived from the `Image` control class and both inherit the `ImageUrl`, `AlternateText`, `DescriptionUrl`, `ImageAlign`, and `GenerateEmptyAlternateText` properties from the `Image` class.

An `ImageButton` control is used to show a clickable image and can be used for postback. The `ImageButton` control causes postback when the user clicks anywhere in the control. This class is derived from the `Image` class and implements the `IPostBackDataHandler`, `IPostBackEventHandler`, and `IButtonControl` interfaces.

The `ImageMap` consists of an image that needs to be displayed with clickable areas defined by x and y coordinates known as hotspots. The x and y coordinates for the clickable areas have to be defined for the location and size of the hotspot. An `ImageMap` control can contain any number of hotspots. It is not mandatory for the entire graphic to be filled with hotspots as well. Every hotspot can trigger a postback event or open a different link.

Using the HotSpot Control

The `HotSpot` control is available in ASP.NET 2.0 and newer versions. You can use the `HotSpot` control to define clickable areas called hotspots to perform different actions.

For example, a web application on human health might display images of a human figure with hotspots defined all over the body. By clicking on different parts of the body, users display linked pages that contain additional information. Hotspots may be defined on the home page image of an application as well to facilitate navigation across the Web site. Hotspots can be set in three different shapes: Circular, Rectangular, and Polygon, with the help of the derived classes `CircularHotSpot`, `RectangularHotSpot`, and `PolygonHotSpot`, respectively.

You can also define multiple overlapping hotspots in any specific order. Based on the order in which you defined them, the first hotspot will take precedence over the others.

Some important properties of the `HotSpot` control are:

- **AlternateText:** Specifies the text that will be displayed when the referred image of a hotspot is not available or visible.
- **HotSpotMode:** Indicates the behavior of the hotspot when clicked; it can be set to `NotSet`, `Inactive`, `Navigate`, or `PostBack`. In PostBack mode, a `Click` event results in a postback to the server. In Inactive mode, a `Click` event does not result in anything. In Navigate mode, a `Click` event results in displaying the page set in the `NavigateURL` property. If `HotSpotMode` property as well as `ImageMap` are set, the `HotSpotMode` property takes precedence.
- **Target:** Specifies that the web page will be displayed in a new window.
- **NavigateUrl:** Specifies the URL to navigate to when the hotspot is clicked.

Using the Panel Control

The Panel control is a container that is useful when a group of controls need to be displayed or hidden.

The Panel control is not visible to a user because it has no user interface (UI). It generates the `<asp:Panel>` tag inside, which you can add to your controls. This control renders a `<div>` tag.

The Panel control is derived from the `WebControl` class.

Using the properties of the Panel control, you can specify a URL for the background image to control or align the panel contents horizontally or to enable the wrapping feature or group text. The `BackImageUrl`, `HorizontalAlign`, `Wrap`, `GroupingText` properties are used for each respective task.

You can also use the PlaceHolder control in ASP.NET to add the controls at runtime. The major difference between the PlaceHolder control and the Panel control is that by using the Panel control in addition to adding controls at runtime, you can add controls at design time. Additionally, the Panel control renders a `<div>` tag, but the PlaceHolder doesn't generate any tag.

The following markup illustrates how you can add a Label and a TextBox control in the Panel control:

```
<asp:Panel ID="Panel1" runat="server" BorderStyle="Dotted"
BorderWidth="2px">
<asp:Label ID="Label1" runat="server" Text="Label
Control"></asp:Label>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</asp:Panel>
```

At runtime, the controls are rendered inside the `<div>` tag with a dotted border.

Using the Wizard Control

The Wizard control creates a wizard that captures user input by breaking data into logical steps and presenting them to the end user.

The Wizard control uses steps to define different sections for user input. The steps of the wizard are developed using a series of `WizardStep` controls. Every step is given by a `StepType` property to indicate whether it is a beginning, intermediate, or completion step. Wizards can have multiple intermediate steps. Within each step, you can use different controls, such as `TextBox` and `ListBox`, to collect input from the user. In the completion step of the wizard, all collected data is accessible for further processing or validation.

Using Wizard control, you can customize navigation within the steps without writing code because the Wizard control ensures that data is persistent across pages or steps. The Wizard control has a State Management feature that enables you to build linear and nonlinear navigation. This facilitates forward and backward movement. In addition, it allows selection of individual steps at any point in time as long as the sidebar is displayed.

You can use the `StepNextButtonText`, `StepPreviousButtonText`, and `FinishCompleteButtonText` properties to customize the text for navigation.

The Wizard control displays the current step and its title automatically. You can use its `HeaderText` property to set the text caption that is displayed on the header area of the control.

The `Wizard` class is derived from the `CompositeControl` class. This class implements the basic functionality needed by the web server control that contains child controls.

Some important properties of the Wizard control are:

- **ActiveStepIndex:** Specifies the currently active step. The default value is -1.
- **CancelButtonText:** Specifies the text for the Cancel button.

MORE INFORMATION

To know more about Panel control and the properties, refer to the Panel Web Server Control Overview section on MSDN.

- **CancelButtonType:** Specifies the type of the Cancel button. Its values can be Button, Image, or Link.
- **CancelDestinationPageUrl:** Indicates the URL to be redirected to when the Cancel button is clicked.
- **FinishCompleteButtonText:** Specifies the text for the Finish Complete button.
- **FinishCompleteButtonType:** Specifies the type of the Finish Complete button. The values can be Button, Image, or Link.
- **FinishCompleteDestinationPageUrl:** Indicates the URL to be redirected to when the Finish Complete button is clicked.
- **FinishPreviousButtonText:** Specifies the text of the Previous button of the finish step.
- **FinishPreviousButtonType:** Specifies the type of the Previous button of the finish step. The values can be Button, Image, or Link.
- **StepNextButtonText:** Specifies the text for Next button.
- **StepNextButtonType:** Specifies the type of the Next button. The values can be Button, Image, or Link.
- **StepPreviousButtonText:** Specifies the text for the Previous button.
- **StepPreviousButtonType:** Specifies the type of the Previous button. The values can be Button, Image, or Link.

Let's use the Wizard control to create a sample application that captures user information in a step-by-step manner for processing a loan request.

The first step of the wizard shows the Welcome message. In the second step, the wizard asks the user for their name and contact number using TextBox and Label controls. In the third step, the wizard asks the user to select the type of loan they are interested in: Personal Loan, Home Loan, or Vehicle Loan. To do this, you add three RadioButton controls and set the **GroupName** property to **LoanGroup**. The final step displays a message informing the user that their interest has been captured and the company will reach out to them.

To create the application, first add a new web page, **LoanRequest.aspx**, in your application. Then add the Wizard control from the ToolBox. Adding the controls will automatically generate the markup in the .aspx file, you can remove that code and add the following mark-up for the **LoanRequest.aspx** page:

```
<form id="form1" runat="server">

    <asp:Wizard ID="Wizard1" runat="server" ActiveStepIndex="0"
    Height="113px"
        onfinishbuttonclick="Wizard1_FinishButtonClick" Width="565px">
        <WizardSteps>

            <asp:WizardStep runat="server" Title="Welcome">
                <b>Welcome to MyLoan Inc.</b>
            </asp:WizardStep>

            <asp:WizardStep ID="step1" runat="server"
            title="User Information">
                Enter Name:
                <asp:TextBox ID="txtName" runat="server">
                </asp:TextBox>
                <br />
                Contact No:
                <asp:TextBox ID="txtContactNo" runat="server">
                </asp:TextBox>
            </asp:WizardStep>

            <asp:WizardStep ID="step2" runat="server" title="Loan Types">
                Select the Type of Loan you are interested in:<br />
            </asp:WizardStep>
        </WizardSteps>
    </asp:Wizard>
</form>
```

```

<asp:RadioButton ID="rdPersonalLoan" runat="server"
GroupName="LoanGroup" Text="Personal Loan" />
    <asp:RadioButton ID="rdHomeLoan" runat="server"
GroupName="LoanGroup" Text="Home Loan" />
        <asp:RadioButton ID="rdVehicleLoan" runat="server"
GroupName="LoanGroup" Text="Vehicle Loan" />
            </asp:WizardStep>

            <asp:WizardStep ID="Finish" runat="server" StepType="Finish"
Title="Complete">
                Thank you for your interest. We will contact you shortly.
            </asp:WizardStep>
        </WizardSteps>
    </asp:Wizard>
</form>

```

The code-behind file for the Wizard1_FinishButtonClick method is displayed:

```

protected void Wizard1_FinishButtonClick(object sender,
WizardNavigationEventArgs e)
{
    // Add your code for the finish button Click.
}

```

Figures 2-4 through 2-7 shows the steps that are displayed in the designer view.

Figure 2-4

Wizard steps in designer view—the home page

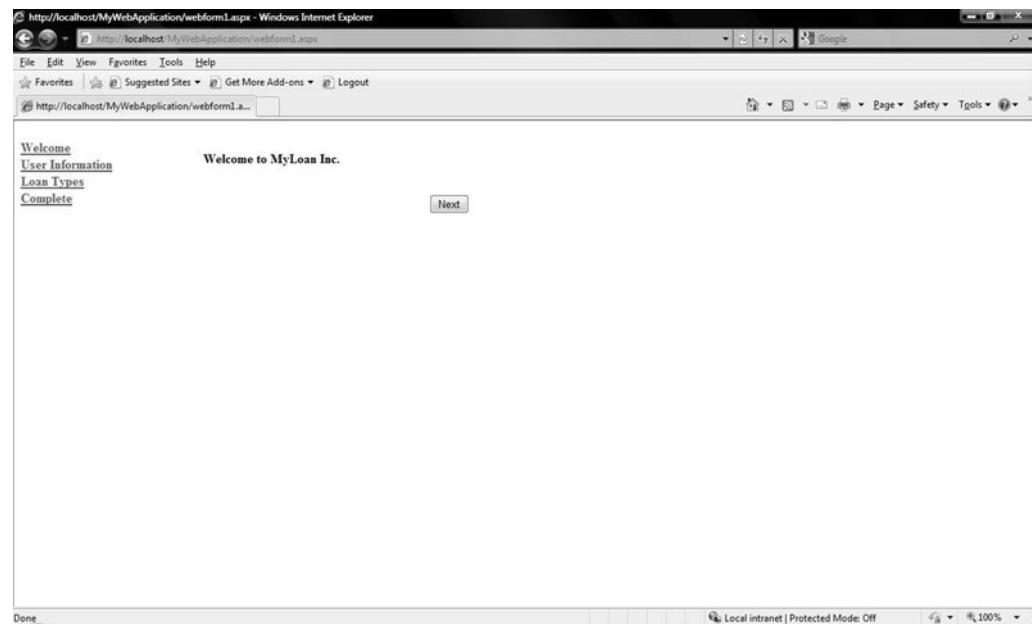
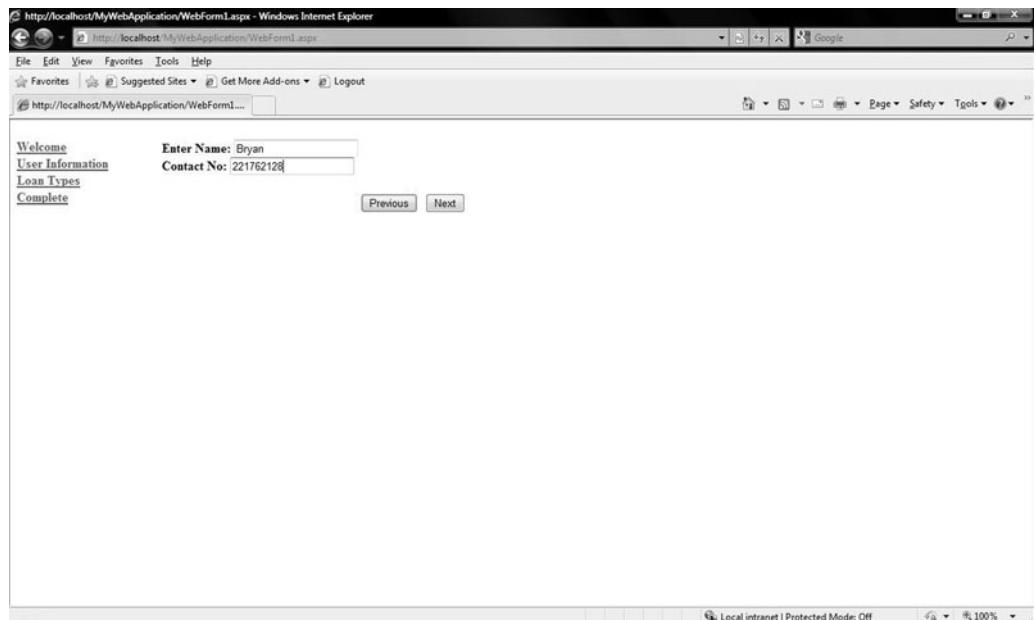


Figure 2-5

Wizard steps in designer view—the user information page

**Figure 2-6**

Wizard steps in designer view—the loan types page

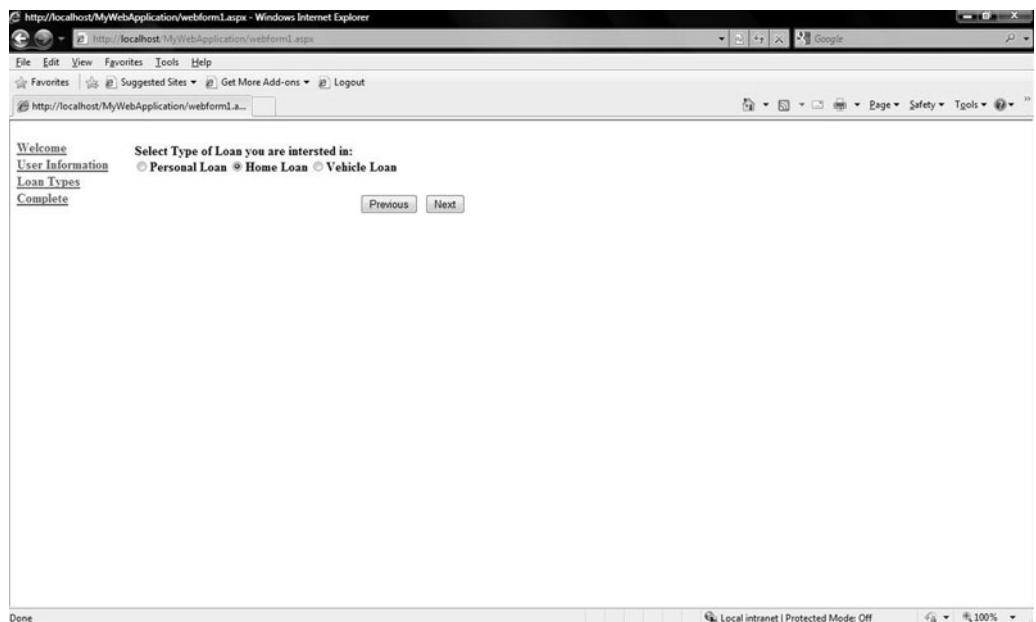
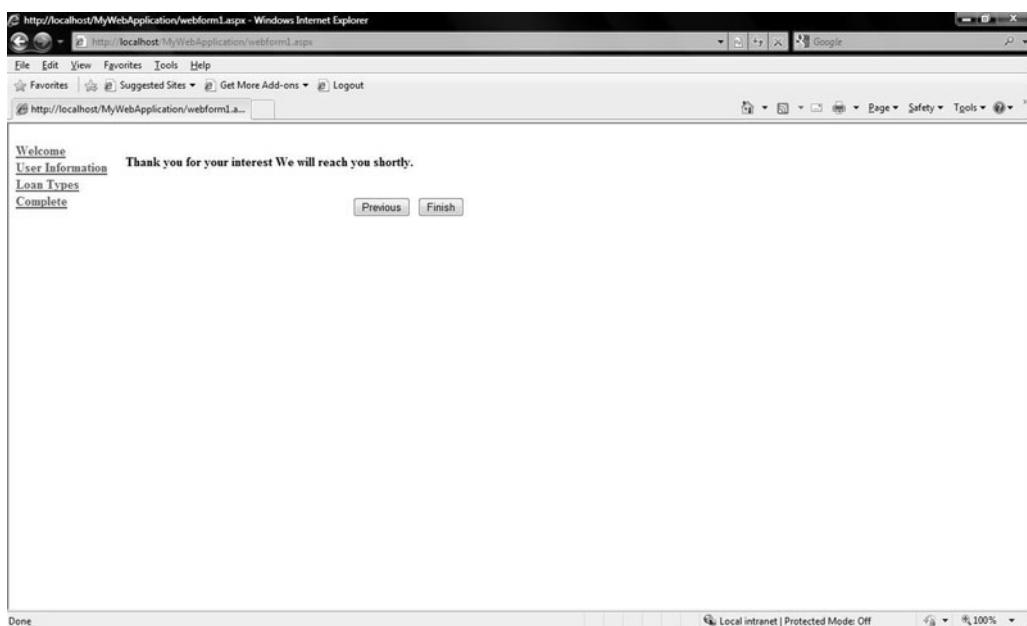


Figure 2-7

Wizard steps in designer view—the exit page

**TAKE NOTE***

The *Wizard* control is a new feature in ASP.NET 2.0 and higher versions. You may find it difficult to use the Wizard control in your applications due to its complicated design.

In the previous example, you used the *Wizard* control and *WizardStep* control classes. Some important classes that relate to the *Wizard* control are:

- **Wizard:** The main class for the control.
- **WizardStep:** Contains the basic steps that are displayed in a wizard.
- **WizardStepCollection:** The collection of wizard steps that contain the user interface in each step.
- **WizardStepType:** Specifies the types of navigation UI that can be displayed for a step.
- **WizardEventArgs:** Contains the data required for the navigation events.
- **WizardNavigationEventHandler:** The set of navigation events that are triggered on various actions.

Using the FileUpload Control

The *FileUpload* control enables users to upload files to a web application.

This control shows a textbox along with a *Browse* button that users can use to specify the location of the file or to browse to the location on the local machine or a different network. After a file is uploaded, you can save the file elsewhere on a database or the file system as well.

The *FileUpload* control allows you to limit the size of the file that is being uploaded and also examine the properties of that file.

The file uploaded using this control is sent as part of an HTTP request and is cached in the server memory. You can access the uploaded file using the following properties:

- **FileBytes:** Allows access to the uploaded file as a byte array
- **FileContent:** Allows access to the uploaded file as a stream
- **PostedFile:** Allows access to the uploaded file as an object of the *HttpPostedFile* class. This class exposes the *ContentType* and *ContentLength* properties.

To add the *FileUpload* control to your application, first add a new web form *FileUploadTest* in the application, and then drag and drop the *FileUpload* control and a button. On the *Click* event handler of the button, add the following code snippet:

```
protected void Button1_Click(object sender, EventArgs e)
{
    if (FileUpload1.HasFile)
    {
```

```

        Response.Write("File name = "+ FileUpload1.FileName);
        Response.Write("<br/>Length = "+ FileUpload1.FileBytes
.Length.ToString());
        Response.Write("<br/>Type = "+ FileUpload1.PostedFile
.ContentType);
    }
}

```



WARNING While providing the FileUpload functionality in your web application, it is possible that users might not always upload safe and secure files. It is imperative that you check the characteristics of the file after it gets uploaded.

Run the application, select the file using the browse button, and click the button. The file name, the size of file in bytes, and the MIME type of the file will be displayed.

The uploaded file can be saved on the server's hard disk if the ASP.NET process has the necessary permissions to add files at the specified location.

Using the MultiView Control

You can use the MultiView control to show or hide different sections of a web page.

A tabbed page requirement or a long form with multiple sections is a good example of a scenario where you would use a MultiView control. For instance, a web application that hosts employee information, from personal to education to professional background, is best suited for a MultiView control. A MultiView control consists of multiple View controls. You can use the MultiView control at any instance to select a particular view and render it to the web page. A Wizard control provides more navigation features than a MultiView control; you can choose whether you want to submit the page after each step or after the final step.

The `ActiveViewIndex` property specifies the View control that needs to be displayed using the index value. The `Views` property allows you to retrieve the collection of the view controls contained in the MultiView control.

In addition to these properties, the `GetActiveView` and `SetActiveVIew` are the two methods that retrieve the view control selected and help select the active view.

TAKE NOTE*

MultiView and View controls are new features in ASP.NET 2.0 and higher versions.

CERTIFICATION READY?

Consume standard controls.

2.5

■ Creating Data-Bound Controls



THE BOTTOM LINE

You use data-bound controls to display data from various data sources, such as databases and XML files, or from any other data source in different formats on a web page.

Suppose you want to create a web application for a university. The application should store information about courses, students, lecturers, and departments of the university. You want to show the list of departments and the students enrolled in a particular course to the users. To display data on a web page, you first need to retrieve the data from data sources and create a collection to arrange it. Then you need to bind the arranged collection to the `DataSource` property of the control for display on the web page.

Creating a Data-Binding Collection

You want to retrieve the list of departments from the data source. The following code shows how the `GetDepartments` method returns a list of departments in the university.

The first step adds the `Department` class definition as shown in this code:

```

public class Department
{
    public string DepartmentName

```

```

    { get; set; }
    public int StudentCount
    { get; set; }
}

```

Now let's create a method that returns the list of departments as shown:

```

Private List<Department> GetDepartments()
{
    List<Department> dList = new List<Department>();
    dList.Add(new Department() {DepartmentName="Arts",
StudentCount=2000 });
    dList.Add(new Department() {DepartmentName ="Science",
StudentCount=13130 });
    dList.Add(new Department() {DepartmentName ="Management",
StudentCount=7106 });
    dList.Add(new Department() {DepartmentName = "Law",
StudentCount=1202 });
    return dList;
}

```

The calling method can now bind the collection of departments, `List<Department>`, to the data-bound control. End users can also update information using the data-bound controls, which can later be saved back to the data source.

Binding Data to Data-Bound Controls

After retrieving data in the collection, you need to bind it to the data-bound control to display information to the user. To bind the `List<Department>` collection to the data-bound control, add the following code in the `Page_Load` method:

```

Protected void Page_Load(object sender, EventArgs e)
{
    DataGrid dg = new DataGrid();
    dg.Caption = "List of Departments";
    dg.DataSource = GetDepartments();
    dg.DataBind();
    Form.Controls.Add(dg);
}

```

TAKE NOTE*

The `.DataBind` method is called on child controls recursively. If the `.DataBind` method is called on a page, it will in turn call the `.DataBind` methods of all the controls within the page.

This example uses a `DataGrid` control to display data in the tabular manner. Note that the collection of departments is bound to the `DataSource` property of the `DataGrid` control to display data on the web page. The `.DataBind` method is called when the data is ready to be read from the source.

There are three categories of data-bound controls:

- **Simple:** Includes the `ListControl` and `AdRotator` controls and all other controls that are derived from these controls.
- **Composite:** Includes the `GridView`, `DetailsView`, and `FormsView` controls, which are derived from the `CompositeDataBoundControl` class.
- **Hierarchical:** Includes `CompositeMenu` and `TreeView` controls that are derived from the `HierarchicalDataBoundControl` class, which in turn, is derived from the `BaseDataBoundControl` class.

The `DataBoundControl` class is derived from the `BaseDataBoundControl` class, and its parent classes are `CompositeDataBoundControl` and `ListControl`. The `DataMember` string property of the `DataBoundControl` class is used when the data source has more than one result set.

CERTIFICATION READY?

Bind controls to data by using data-binding syntax.

Using the ListView Control

The ListView control uses user-defined templates to display the value of a data source.

Using a ListView control, you can perform operations on records such as select, edit, insert, and delete.

Following is the markup that shows a list of students using a ListView control:

```
<asp:ListView ID="StudentListView" DataSourceID="StudentDataSource"
DataKeyNames="StudentID" runat="server">
    <LayoutTemplate>
        <table cellpadding="2" width="640px" border="1" ID="StudentTable"
runat="server">
            <tr id="Tr1" runat="server" style="background-color: #98FB98">
                <th id="Th2" runat="server">Student Name</th>
                <th id="Th3" runat="server">Age</th>
                <th id="Th4" runat="server">Stream</th>
            </tr>
            <tr runat="server" id="itemPlaceholder" />
        </table>
    </LayoutTemplate>
    <ItemTemplate>
        <tr id="Tr2" runat="server">
            <td>
                <asp:Label ID="StudentNameLabel" runat="server" Text='<%#
Eval("StudentName") %>' /></td>
                <td><asp:Label ID="AgeLabel" runat="server" Text='<%#
Eval("Age") %>' /></td>
                <td> <asp:Label ID="StreamLabel" runat="server" Text='<%#
Eval("Stream") %>' /></td>
            </tr>
    </ItemTemplate>
</asp:ListView>
```

In this code, you can add the LinqDataSource with id StudentDataSource and bind it to the database in the .aspx page.

Understanding the DataSource Objects

DataSource and DataSourceID are important properties of the `BaseDataBoundControl` class. The `DataSource` property gets or sets the object that the data-bound control uses to get the list of data items. The `DataSource` property is used to bind to any object that implements the `IListSource` or `IEnumerable` interface. For example, a `DataGridView` control can bind directly to a `DataTable` or `DataView` (`IListSource`) or to various `List` objects (`IEnumerable`).

You can set both `DataSource` or `DataSourceID` properties for a control, but in this case, `DataSourceID` gets precedence.

A GUI-based data source control provides a consistent way to bind data sources to a control. These controls are derived from the `DataSourceControl` or `HierarchicalDataSourceControl` classes and implement the interfaces `IDataSource` and `IListSource`.

Some important types of GUI-based data source controls are:

- **AccessDataSource:** Binds to Microsoft Access database files with the .mdb extension.
- **SqlDataSource:** Binds to ODBC, OLEDB, SQL Server, Oracle, and other databases that use SQL.

X REF

You will learn more about data sources in Lesson 4.

- **XmlDataSource:** Binds to XML files with support for transformation and Xpath expressions.
- **ObjectDataSource:** Binds to a middle-tier business object. An instance of the class is created when data is required. You must select the methods for insert, update, and delete operations. The `Select` method returns a single object that implements the interfaces `IEnumerable`, `IListSource`, `IDataSource`, and `IHierarchicalDataSource`. This indicates that `DataTable`, `DataSet` or collection objects can be used in the data source object.
- **SiteMapDataSource:** Connects to a site navigation tree for the application, provided that a valid sitemap file is present at the application root.
- **LinqDataSource:** Allows you to use language-integrated queries (LINQ) to retrieve and modify the data in the database.

Using the DataBinder Class

The `DataBinder` class supports Rapid Application Development (RAD). You can generate and parse data-binding expression syntax using this sealed class.

RAD designers can make use of the `Eval` method that uses reflection to parse and evaluate a data-binding expression against an object.

The following line of code demonstrates how the `Eval` method can be used declaratively to bind to a `Name` field. This example uses container syntax using one of the listed web server controls:

```
<%# DataBinder.Eval (Container.DataItem, "Name") %>
```

Suppose you want to display the list of university departments in a drop-down list with the help of the `DataBinder` class. To do this, write the following snippet that uses the `Eval` method of the `DataBinder` class:

```
<form id="form1" runat="server">
    <asp:DataList id="DepartmentList" BorderColor="black"
        CellPadding="5" CellSpacing="5" RepeatDirection="Vertical"
        RepeatLayout="Table" RepeatColumns="3" runat="server">
        <HeaderStyle BackColor="#aaaadd">
        </HeaderStyle>
        <AlternatingItemStyle BackColor="Gainsboro">
        </AlternatingItemStyle>
        <HeaderTemplate>
            List of Departments:
        </HeaderTemplate>
        <ItemTemplate>
            Department: <br />
            <%# DataBinder.Eval(Container.DataItem, "StringValue") %>
            <br />
        </ItemTemplate>
    </asp:DataList>
</form>
```

In this code, `DataList` is a data-bound control that displays a list of items using the item template.

You can add the earlier markup in the .aspx page to ensure that you've provided the correct implementation for the data binding. To do this, you can add the following snippet inside the `HEAD` tag:

```
<script runat="server">
    ICollection CreateDataSource()
```

```

{
    DataTable dt = new DataTable();
    DataRow dr;
    dt.Columns.Add(new DataColumn("StringValue", typeof(String)));
    dr = dt.NewRow();
    dr[0] = "Science";
    dt.Rows.Add(dr);
    dr = dt.NewRow();
    dr[0] = "Finance";
    dt.Rows.Add(dr);
    DataView dv = new DataView(dt);
    return dv;
}
void Page_Load(Object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        DepartmentList.DataSource = CreateDataSource();
        DepartmentList.DataBind();
    }
}
</script>

```

Using Template Controls

To separate content from presentation, ASP.NET provides templates that can be applied to controls. These templated controls do not provide a user interface, instead they act as a container to include a class whose properties and methods are accessible to the host page.

The seven types of template controls available are:

- **HeaderTemplate:** Defines the contents to be displayed in the header row of the control.
- **FooterTemplate:** Defines the contents to be displayed in the footer row of the Control.
- **ItemTemplate:** Defines the contents for the control's row when it uses read-only mode.
- **InsertItemTemplate:** Defines the contents for the control's row when it uses insert mode. This template provides the input controls and commands for inserting the new records.
- **EditItemTemplate:** Defines the contents for the control's row when it uses edit mode. This template provides the input controls and commands for editing the existing record.
- **EmptyDataTemplate:** Displays alerts if there is no record in the bound column.
- **PagerTemplate:** Defines the content for the pager row when the `AllowPaging` property is `true`. This template provides the controls that users can utilize to navigate to another record.

The following markup shows you how to apply the HeaderTemplate and ItemTemplate on the DataList control:

```

<asp:DataList ID="StudentList" runat="server">
    <HeaderTemplate>Student List</HeaderTemplate>
    <ItemTemplate>
        <asp:label id="Label1" runat="server" Text='<%# Eval("DataItem.StudentName")%>'></asp:label>
    </ItemTemplate>
</asp:DataList>

```

Note that you need to bind data to the data source to make this snippet functional.

Using List Controls

You can use list controls to display data in the form of a list.

There are various types of list controls, such as ListControl, DropDownList, ListBox, CheckBoxList, and BulletedList.

ListControl is an abstract base class inherited by other classes such as CheckBoxList, DropDownList, ListBox, and RadioButtonList.

The ListControl class provides the following properties:

- **DataSource:** Attaches a data source to the control.
- **DataMember:** Specifies the table in case the data source has multiple tables.

There are other properties, such as DataTextField and DataValueField properties, which you can use to bind fields to the data sources.

The following code snippet will help you understand the Text and Value properties of ListItem. At runtime, the user will be able to see the departments such as Arts, Management, and Law. When the user selects any item and postback occurs, values such as ARTS and LAW are sent to the server:

```
<asp:ListBox ID="DepartmentList" runat="server">
    <asp:ListItem Value="ARTS">ARTS Department</asp:ListItem>
    <asp:ListItem Value="MANAGEMENT">MANAGEMENT
    Department</asp:ListItem>
    <asp:ListItem Value="LAW">LAW Department</asp:ListItem>
    <asp:ListItem Value="Science">SCIENCE Department</asp:ListItem>
</asp:ListBox>
```

List Items that are declared programmatically can be added using the AppendDataBoundItems property with their default value set to false. Whenever the selection in the list controls is changed, the SelectedIndexChanged event is fired.

In a list, you can obtain the index and the property of the selected item by using the SelectedIndex and the SelectedItem properties.

In this sample application, to let the user navigate to a web page when they click on a department, the SelectedIndexChanged event will contain the code:

```
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (ListBox1.SelectedValue == "ARTS")
    {
        // Open the ARTS department page.
        Response.Redirect("ARTS.aspx");
    }
    else
    {
        // Display the site is under construction.
        Response.Redirect("Construction.aspx");
    }
}
```

The CheckBoxList control allows you to select more than one check box to be selected in a group of check boxes that are dynamically generated with data binding. To determine which items are selected, the Items collection needs to be iterated through for Selected property of all items in the collection.

To control the direction and layout of the list display, the RepeatLayout and RepeatDirection property settings can be used:

- **RepeatLayout.Table (default):** Renders the list within a table.
- **RepeatLayout.Flow:** Renders the list without any table structure.

- **RepeatDirection.Vertical**: Renders the list vertically.
- **RepeatDirection.Horizontal**: Renders the list horizontally.

The ListBox and the DropDownList controls derive from the `ListControl` class. Both of these controls use the `Items` collection to store all the objects contained in the list.

The RadioButtonList control is similar to the CheckBoxList control. The only difference is that when getting the selected value from a RadioButtonList control, you need to use the `SelectedValue` property instead of the `Selected` property.

The BulletedList control renders a list of items in a bulleted format, such as bullets or numbers. The bullets can be rendered as a custom image, circle, square, or disc-shaped symbol by using the `BulletStyle` property.

The BulletedList control can display as a Text, Hyperlink, or Link button by setting its `DisplayMode` property.

Using Composite Data-Bound Controls and Hierarchical Data-Bound Controls

Suppose you want to display the department-wise hierarchy of the university staff, or you want to display detailed information about the courses offered through the Information Technology section, which is part of the Science department. In these scenarios, composite data-bound controls and hierarchical data-bound controls are very useful.

`CompositeDataBoundControl` is a base class for the controls that contains other data-bound controls. This class implements the `INamingContainer` interface, which means the derived class can be the naming container for the child controls. The `FormsView`, `DetailsView`, and `GridView` controls are derived from the `CompositeDataBoundControl` class.

The `HierarchicalDataBoundControl` is a base class for controls that render data in a hierarchical fashion. The `TreeView` and `Menu` controls are derived from the `HierarchicalDataBoundControl` class.

Using the GridView Control

The `GridView` control displays data in a tabular format and provides the ability to perform sorting, paging, and handling postbacks.

This control is similar to the `DataGrid` control. It allows you to display data in the form of rows and columns using the `GridViewRow` and `DataColumnField` objects. This control can display check boxes, buttons, commands, hyperlinks, images, and templates. Some important properties of the `GridView` control are:

- **AllowPaging**: Contains a Boolean value indicating whether paging is turned on. The default value is `false`.
- **AllowSorting**: Contains a Boolean value indicating whether a column header can be used to sort the data. The default value is `false`.
- **AutoGenerateDeleteButton**: Contains a Boolean value indicating whether a Delete button is automatically generated. The default value is `false`.
- **AutoGenerateEditButton**: Contains a Boolean value indicating whether an Edit button is automatically generated. The default value is `false`.
- **AutoGenerateSelectButton**: Contains a Boolean value indicating whether a Select button is automatically generated. The default value is `false`.
- **EnableSortingAndPagingCallbacks**: Contains a Boolean value indicating whether a client script for paging and sorting should be rendered to the browser that supports callbacks.
- **EmptyDataText**: Represents the text displayed in the empty data row when the `EmptyDataTemplate` is defined.

- **EnableModelValidation:** Contains a Boolean value indicating whether page validation will be performed after validation is complete in the model.

The following properties of the GridView control enable you to customize its appearance:

- **PagerStyle:** Sets the appearance of a pager row.
- **SelectedRowStyle:** Sets the appearance of a selected row.
- **HeaderStyle:** Sets the appearance of the GridView header.
- **FooterStyle:** Sets the appearance of the GridView footer.
- **EmptyDataRowStyle:** Sets the appearance of an empty row.
- **EditDataRowStyle:** Sets the appearance of a row selected for editing.

Using the AdRotator Control

The AdRotator control is used to display selected advertising banners in a random manner on a web page. This control is derived from the **DataBoundControl** class.

When this control is rendered to HTML, it generates the `<a>` and `` elements. Advertising data can be populated from an XML file or from a database.

Here are some important properties of the AdRotator control:

- **AdvertisementFile:** Specifies the location of the XML file that contains the advertisements.
- **KeywordFilter:** Specifies the keyword for filtering advertisements.

Following are the important attributes of the XML advertising file:

- **Keyword:** Indicates the category key of the advertisement, which can be used for filtering the advertisements.
- **ImageUrl:** Indicates the URL for the advertisement image.
- **NavigateUrl:** Indicates the URL to be navigated to when a user clicks on the advertisement.
- **Impression:** Indicates the number used to measure the frequency of the advertisement being displayed. The sum of all the advertisements must be less than 2,048,000,000.

Here are some guidelines that you need to follow:

- Use the App_Data folder to store the advertisement files because browsers do not have access to this folder.
- Consider using file extensions; for example, you can use the `.config` extension because ASP.NET does not allow browsers to request this extension.
- Set the permissions on the advertisement file to allow the ASP.NET account to have read-only access.

Always check the data in the advertisement files for any malicious scripts because AdRotator does not perform this validation.

Using the DetailsView Control

The DetailsView control gives you the ability to render a single record at a time from a data source.

This is useful in a master-details scenario where selecting the master row should display the details of the row. In addition to displaying details, the DetailsView control can be extended to update or delete records and it supports paging using client-side callbacks through its **EnablePagingCallbacks** property.

For example, suppose you want to display the list of all the departments in the university, and when the user selects one department, you want to display the details of that department. To do this, you need to use the GridView control to display the list of departments and the DetailsView control to display the details of the selected department.

Here is the markup for the GridView in the .aspx page:

```
<asp:GridView ID="gvSelectDept" runat="server" CellPadding="4"
    ForeColor="#333333" GridLines="None" AutoGenerateColumns="False"
    OnRowCommand="gvSelectDept_RowCommand" >
    <FooterStyle BackColor="#1C5E55" Font-Bold="True"
        ForeColor="White" />
    <RowStyle BackColor="#E3EAEB" />
    <EditRowStyle BackColor="#7C6F57" />
    <SelectedRowStyle BackColor="#C5BBAF" Font-Bold="True"
        ForeColor="#333333" />
    <PagerStyle BackColor="#666666" ForeColor="White"
        HorizontalAlign="Center" />
    <HeaderStyle BackColor="#1C5E55" Font-Bold="True"
        ForeColor="White" />
    <AlternatingRowStyle BackColor="White" />
    <Columns>
        <asp:BoundField DataField="DepartmentID" Visible="False" />
        <asp:BoundField DataField="DepartmentName"
            HeaderText="DepartmentName" />
        <asp:CommandFieldButtonType="Button" SelectText="View Details"
            ShowSelectButton="True" />
    </Columns>
</asp:GridView>
<asp:DetailsView ID="dvShowDetails" runat="server"
    Height="50px" Width="125px" CellPadding="4"
    ForeColor="#333333" GridLines="None">
    <FooterStyle BackColor="#507CD1" Font-Bold="True"
        ForeColor="White" />
    <CommandRowStyle BackColor="#D1DDF1" Font-Bold="True" />
    <EditRowStyle BackColor="#2461BF" />
    <RowStyle BackColor="#EFF3FB" />
    <PagerStyle BackColor="#2461BF" ForeColor="White"
        HorizontalAlign="Center" />
    <FieldHeaderStyle BackColor="#DEE8F5" Font-Bold="True" />
    <HeaderStyle BackColor="#507CD1" Font-Bold="True"
        ForeColor="White" />
    <AlternatingRowStyle BackColor="White" />
</asp:DetailsView>
```

In the Page_Load method, bind the GridView to the list of departments as shown:

```
protected void Page_Load(object sender, EventArgs e)
{
    gvSelectDept.DataSource = GetDepartments();
    // GetDepartments will return the list of departments.
    gvSelectDept.DataBind();
}
```

Add the event handler for the RowCommand event for the GridView control where you'll be adding the code to display the details as shown:

```
protected void gvSelectDept_RowCommand(object sender,
    GridViewCommandEventArgs e)
{
    // TODO: Add your code to display the courses offered.
    dvShowDetails.DataSource = GetInfo();
    dvShowDetails.DataBind();
}
private List<Course> GetInfo()
```

```

    {
        List<Course> c = new List<Course>();
        c.Add(new Course() { CourseID = 1, DepartmentID = 1,
        CourseName = "Master in Information Technology" });
        return c;
    }
    public class Course
    {
        public int CourseID
        { get; set; }
        public int DepartmentID
        { get; set; }
        public string CourseName
        {get; set;}
    }
}

```

The DetailsView control supports the same formatting as the GridView control with the same set of properties. However, unlike GridView, the DetailsView control does not support sorting.

Using the FormView Control

The FormView control is similar to the DetailsView control in that it displays data records one at a time from the source. The major difference between FormView and DetailsView is that DetailsView has a built-in tabular rendering, whereas the FormView control needs a user-defined template for its rendering.

Let's create a web page that will show a list of vehicles using the FormView control and a SqlDataSource:

```

<form id="form1" runat="server">
<div>
<asp:SqlDataSource id="SqlDataSource1"
selectcommand="Select [Vin], [Make], [Model], [Year], [Price] From
[Vehicles]" connectionstring="<%$.ConnectionStrings:NorthWindConnectionString%>" runat="server"/>
<br />
<br />
    &nbsp; &nbsp; &nbsp; &nbsp;
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click"
Style="z-index: 101; left: 20px; position: absolute; top: 45px"
Text="Load Vehicles" />

<asp:FormView ID="FormView1" runat="server" AllowPaging="True"
DataKeyNames="Vin" DataSourceID="SqlDataSource1" Width="100%">
<ItemTemplate>
    <table>
        <tr>
            <td align="center">
                <hr />
                <span style="font-weight: bold;
color: blue">VIN:</span>&nbsp;
                <asp:Label ID="VinLabel" Width="105px" runat="server"
Text='<%# Eval("Vin") %>'> </asp:Label>&nbsp;&nbsp;
                <span style="font-weight: bold; color: blue">Make:</
span>&nbsp;
                <asp:Label ID="MakeLabel" Width="105px" runat="server"
Text='<%# Eval("Make") %>'> </asp:Label>&nbsp;&nbsp;
                <span style="font-weight: bold; color:
blue">Model:</span>&nbsp;
            </td>
        </tr>
    </table>
</ItemTemplate>
<FooterTemplate>
    <hr />
    <asp:Label ID="Label1" runat="server" Text="Total Number of Vehicles:</asp:Label>
    <asp:Label ID="Label2" runat="server" Text='<%# Eval("Count") %>'>
</FooterTemplate>
</asp:FormView>

```

```
<asp:Label ID="ModelLabel" Width="105px" runat="server"
Text='<%# Eval("Model") %>'> </asp:Label>&nbsp;&nbsp;
    <span style="font-weight: bold; color: blue">Year:</
span>&nbsp;
    <asp:Label ID="YearLabel" Width="105px" runat="server"
Text='<%# Eval("Year") %>'> </asp:Label><br />
    </td>
</tr>
<tr>
    <td align="center">
        <span style="font-weight: bold; font-size:
x-large; color: blue">Price: </span>
        <span style="font-weight: bold; font-size:
x-large">&nbsp;
            <asp:Label ID="PriceLabel" Width="105px" runat="server"
Text='<%# Eval("Price", "{0:C}") %>'> </asp:Label></span>
        </td>
    </tr>
    <tr>
        <td align="center">
            <hr />
            <asp:LinkButton ID="LinkButton1" runat="server"
CausesValidation="False" CommandName="Edit" Text="Edit">
</asp:LinkButton>
            <asp:LinkButton ID="LinkButton2" runat="server"
CausesValidation="False" CommandName="New" Text="New"> </asp:
LinkButton><asp:LinkButton ID="LinkButton3" runat="server"
CausesValidation="False" CommandName="Delete" Text="Delete">
            </asp:LinkButton>
        </td>
    </tr>
</table>
</ItemTemplate>

<EditItemTemplate>
    <table>
        <tr>
            <td align="center">
                <hr />
                <span style="font-weight: bold; color:
blue">VIN:</span>&nbsp;
                    <asp:Label ID="VinLabel" Width="105px" runat="server"
Text='<%# Eval("Vin") %>'> </asp:Label>&nbsp;&nbsp;
                    <span style="font-weight: bold; color:
blue">Make:</span>&nbsp;
                        <asp:TextBox ID="EditMakeTextBox" Width="100px"
runat="server" Text='<%# Bind("Make") %>'> </asp:TextBox>&nbsp;&nbsp;
                        <span style="font-weight: bold; color:
blue">Model:</span>&nbsp;
                            <asp:TextBox ID="EditModeTextBox" Width="100px"
runat="server" Text='<%# Bind("Model") %>'> </asp:TextBox>&nbsp;&nbsp;
                            <span style="font-weight: bold; color:
blue">Year:</span>&nbsp;
                                <asp:TextBox ID="EditYearTextBox" Width="100px"
runat="server" Text='<%# Bind("Year") %>'>
</asp:TextBox><br />
                                &nbsp;&nbsp;
```

```

        </td>
    </tr>
    <tr>
        <td align="center">
            <span style="font-weight: bold; font-size: large; color: blue">Price: </span>
            <span style="font-weight: bold; font-size: large">&nbsp;
                <asp:TextBox ID="EditPriceTextBox" Width="100px"
runat="server" Text='<%# Bind("Price") %>' > </asp:TextBox></span>
            </td>
        </tr>
        <tr>
            <td align="center">
                <hr />
                <asp:LinkButton ID="LinkButton1" runat="server"
CausesValidation="true " CommandName="Update" Text="Update">
</asp:LinkButton>
            </td>
        </tr>
    </table>
</EditItemTemplate>
</asp:FormView>
</form>
</body>
</html>

```

Note that Eval is a shortcut for DataBinder.Eval while #% casts the expression into string value. <%# %> is the standard data-binding syntax. In the Click event handler of the button, you need to bind the FormView control as shown:

```

VehicleList.Initialize();
FormView1.DataBind();

```

Using the TreeView Control

The TreeView control is derived from the `HierarchicalDataSource` class and displays data in a tree format.

The tree consists of one or more nodes represented by a `TreeNode` object. There are three different types of nodes in a TreeView:

- **Root:** Has no parent node but has one or more child nodes.
- **Parent:** Has one or more child nodes and a single parent node.
- **Leaf:** Has no child nodes.

Any data source that implements the `IHierarchicalDataSource` interface, such as `SiteMapDataSource`, `DataSet`, `XmlDocument`, can be used to populate a TreeView control.

When you create an XML file, say `DepartmentDetails`, the file should look like this:

```

<?xml version="1.0" encoding="utf-8" ?>
<University>
    <Department>Arts</Department>
    <Department>Science</Department>
    <Department>Commerce</Department>
    <Department>Management</Department>
    <Department>Law</Department>
</University>

```

Add a new web form `TreeViewTest.aspx`. Then, from the Toolbox, add the `XMLDataSource` and bind it to `DepartmentDetails.xml`. Now add a `TreeView` and set the `DataSource` property to `XMLDataSource1`.

The markup for the page will be:

```
<form id="form1" runat="server">
    <div>
        <asp:XmlDataSource ID="XmlDataSource1" runat="server"
DataFile="~/UniversityDetails.xml">
            </asp:XmlDataSource>
            <asp:TreeView ID="TreeView1" runat="server"
DataSourceID="XmlDataSource1" Height="126px" Width="193px">
                </asp:TreeView>
            </div>
    </form>
```

Note that this will bind the XML file with the control. You need to add your own code for the Click event for the nodes of the TreeView.

Using the Menu Control

A Menu control provides navigational functionalities to web pages.

Menu controls support information to be rendered at multiple levels in the main menu and submenu forms. For example, the `SiteMapDataSource` class can be used in combination with the Menu control to help users navigate through a Web site.

You can define the bindings between the data item and the `TreeNodes` using the `DataBindings` property of the Menu control. The property is a collection of `MenuItemBindings`.

Suppose you want to display a menu representing a family hierarchy. To do this:

1. Create and add a new XML file named `MyFamily` to your project as shown:

```
<?xml version="1.0" encoding="utf-8" ?>
<Family>
    <Parent>
        id="GrandFather"
        <Parent>
            id="Father"
            <Parent>
                id="Brother"
            </Parent>
            <Parent>
                id="me"
                <child>id="daughter"</child>
            </Parent>
        </Parent>
    </Family>
```

2. Add a Menu control from the Toolbox to your web page.
3. Select the `DataSource` property.
4. Select Add New to open the New dialog box.
5. Select the XML Data Source option and click OK.
6. Specify the name of the `MyFamily.xml` file in the new dialog box and click OK. This binds the Menu control to the XML file.

On running the application, it shows the `Family` tag. When the user moves the mouse to the text `Family`, the menu expands automatically. When the menu item is clicked, nothing happens. This is because you need to implement the `Click` handler logic for the menu item.

This example shows static data binding from an XML file. To populate the Menu control with dynamic binding, you need to retrieve data at runtime from the relational database or an XML file.

Using the DataGrid Control

Suppose you want to display a list of courses offered by the Management department in tabular format. To do this, you can use the DataGrid Control. First, create a web page called CourseInformation.aspx, and then add a DataGrid Control to the page:

```
<div>
    <asp:DataGrid runat="server" ID="DataGrid1"
        AllowPaging="True" AllowSorting="True" PageSize="5">
    </asp:DataGrid>
</div>
```

In the Page_Load method, add the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
    DataGrid1.DataSource = GetCourses();
    DataGrid1.DataBind();
}
```

Note that the `GetCourses` method will return the collection of courses, and so it must be a type that implements `IEnumerable`. When you run this page, it'll show the table with the courses offered. If you click on the column header to sort the information, nothing will happen. To implement sorting, you must provide the implementation in the `SortCommand` method. Similarly, you must provide implementation in the `PageIndexChanged` event to make paging functional.

Some important properties of the DataGrid control are:

- **AllowPaging:** Indicates whether paging is allowed or not. The default value for this property is `false`.
- **AllowSorting:** Indicates whether column headers can be used to sort data. The default value for this property is `false`.
- **AutoGenerateColumns:** Indicates whether to generate the columns automatically depending on the associated data source. The default value is `true`.
- **Caption:** Description for the control.
- **ShowHeader:** Indicates whether to display the header for the control. The default value for this property is `true`.
- **ShowFooter:** Indicates whether to display the footer for the control. The default value for this property is `true`.
- **PageSize:** Indicates the number of items to be displayed per page.
- **AlternatingItemStyle:** Indicates the style applied to alternate items. This property has attributes such as `Font` and `BackColor`.

Using the DataList Control

The DataList control is used to display items using templates.

Some important properties of the DataList control are:

- **AlternateItemStyle:** Indicates the style applied to the alternate items.
- **Caption:** Describes the control.
- **EditItemStyle:** Indicates the style applied to the items in the edit mode.
- **HeaderStyle:** Indicates the style applied to the header.
- **ItemStyle:** Indicates the style applied to the item.
- **RepeatColumns:** Indicates the number of the columns to be used for the layout.
- **RepeatDirection:** Indicates the direction in which items are laid out.

- **RepeatLayout:** Indicates whether items are repeated in a table or in flow. This property controls the layout for the **DownList** control.
- **SelectedItemStyle:** Indicates the style applied to selected items.
- **SeparatedStyle:** Indicates the style applied to the separator items.
- **ShowFooter:** Indicates whether to display the footer. The default value for this property is **true**.
- **ShowHeader:** Indicates whether to display the header. The default value for this property is **true**.

Let's create a sample web page that displays item names, their description, and price. To do this, you first add the **DownList** control on your web page and set the ID to **ItemsList** as shown:

```
<asp:DownList id="ItemsList"
    BorderColor="black" CellPadding="5" CellSpacing="5"
    RepeatDirection="Vertical" RepeatLayout="Table" RepeatColumns="3"
    runat="server">
    <HeaderStyle BackColor="#aaaadd">
    </HeaderStyle>
    <AlternatingItemStyle BackColor="Gainsboro">
    </AlternatingItemStyle>
    <HeaderTemplate>
        List of items
    </HeaderTemplate>
    <ItemTemplate>
        Description: <br />
        <%# DataBinder.Eval(Container.DataItem, "StringValue") %>
        <br />
        Price: <%# DataBinder.Eval(Container.DataItem, "CurrencyValue",
        "{0:c}") %>
    </ItemTemplate>
</asp:DownList>
```

Now you need to add code for data binding in the code-behind file. You can also include the code in the .aspx page by including the **script** tag. To do so, add the following code after the **<Head>** tag in the .aspx page so that you don't have to have a separate code-behind file for this web page:

```
<script runat="server">
    ICollection CreateDataSource()
    {
        // Create sample data for the DownList control.
        DataTable dt = new DataTable();
        DataRow dr;

        // Define the columns of the table.
        dt.Columns.Add(new DataColumn("IntegerValue", typeof(Int32)));
        dt.Columns.Add(new DataColumn("StringValue", typeof(String)));
        dt.Columns.Add(new DataColumn("CurrencyValue", typeof(double)));

        // Populate the table with sample values.
        for (int i = 0; i < 3; i++)
        {
            dr = dt.NewRow();
            dr[0] = i;
```

```

        dr[1] = "Description for item" + (i+1).ToString();
        dr[2] = 1.23 * (i + 1);

        dt.Rows.Add(dr);
    }
    DataView dv = new DataView(dt);
    return dv;
}
void Page_Load(Object sender, EventArgs e)
{
    // Load sample data only once, when the page is first loaded.
    if (!IsPostBack)
    {
        ItemsList.DataSource = CreateDataSource();
        ItemsList.DataBind();
    }
}
</script>

```

You need to import the `System.Data` namespace as follows to make the previous code functional:

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

Notice that in this example, you have used the `DataBinder` class for binding data to the `DownList` Control. You can bind data dynamically at runtime.

Using the Repeater Control

The Repeater control is a data-bound list control that allows custom layout by repeating a specified template for each item in the list.

This control does not have any style or layout. Therefore, as developer, you must explicitly provide the layout, formatting, and style within the control template. The Repeater control does not support editing or selection capability.

If you set the data source for this control that has no records, the control renders the `HeaderTemplate` and `FooterTemplate` without any items. If the data source is set to `null` (Nothing in Visual Basic), the Repeater control does not render.

Important templates that you can use to customize the appearance of the Repeater control are:

- **ItemTemplate:** Defines the content and layout for the items in the list. This is the mandatory template.
- **SeparatorTemplate:** Renders between the items and alternating items. This is an optional template.
- **AlternatingItemTemplate:** Defines the content and layout for the alternating items. This is an optional template.
- **FooterTemplate:** Defines the content and layout for the footer. This is an optional template.
- **HeaderTemplate:** Defines the content and layout for the header. This is an optional template.

The Repeater control class is derived from the `Control` class.

Suppose you want to display the company name and ticker information in a different manner, first in the table and second in a comma-separated list. You can use one instance of the Repeater control to show data in a table and the other one in a comma-separated list as shown in the code snippet for a `.aspx` page:

```

<form id="form1" runat="server">
    <div>
        <b>Repeater1:</b>
        <asp:Repeater ID="Repeater1" runat="server">

```

```
<HeaderTemplate>
    <table border="1">
        <tr>
            <td>
                <b>Company</b>
            </td>
            <td>
                <b>Symbol</b>
            </td>
        </tr>
    </HeaderTemplate>
    <ItemTemplate>
        <tr>
            <td>
                <%# DataBinder.Eval(Container.DataItem, "Name") %>
            </td>
            <td>
                <%# DataBinder.Eval(Container.DataItem, "Ticker") %>
            </td>
        </tr>
    </ItemTemplate>
    <FooterTemplate>
        </table>
        </FooterTemplate>
    </asp:Repeater>
    <br />
    <b>Repeater2:</b>
    <br />
    <asp:Repeater ID="Repeater2" runat="server">
        <HeaderTemplate>
            Company data:
        </HeaderTemplate>
        <ItemTemplate>
            <%# DataBinder.Eval(Container.DataItem, "Name") %>
            (<%# DataBinder.Eval(Container.DataItem, "Ticker") %>)
        </ItemTemplate>
        <SeparatorTemplate>
            ,
        </SeparatorTemplate>
    </asp:Repeater>
</div>
</form>
```

Now you add the script that runs on the server and binds the data to the Repeater control. Add the following code snippet just below the <HTML> tag in the .aspx page:

```
<script language="C#" runat="server">
    void Page_Load(Object Sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            ArrayList values = new ArrayList();
            values.Add(new PositionData("Microsoft Corp.", "Msft"));
            values.Add(new PositionData("Intel Inc.", "Intc"));
            values.Add(new PositionData("Dell", "Dell"));
            values.Add(new PositionData("IBM", "IBM"));
            Repeater1.DataSource = values;
            Repeater1.DataBind();
        }
    }
</script>
```

```

        Repeater2.DataSource = values;
        Repeater2.DataBind();
    }
}
public class PositionData
{
    private string companyname;
    private string ticker;

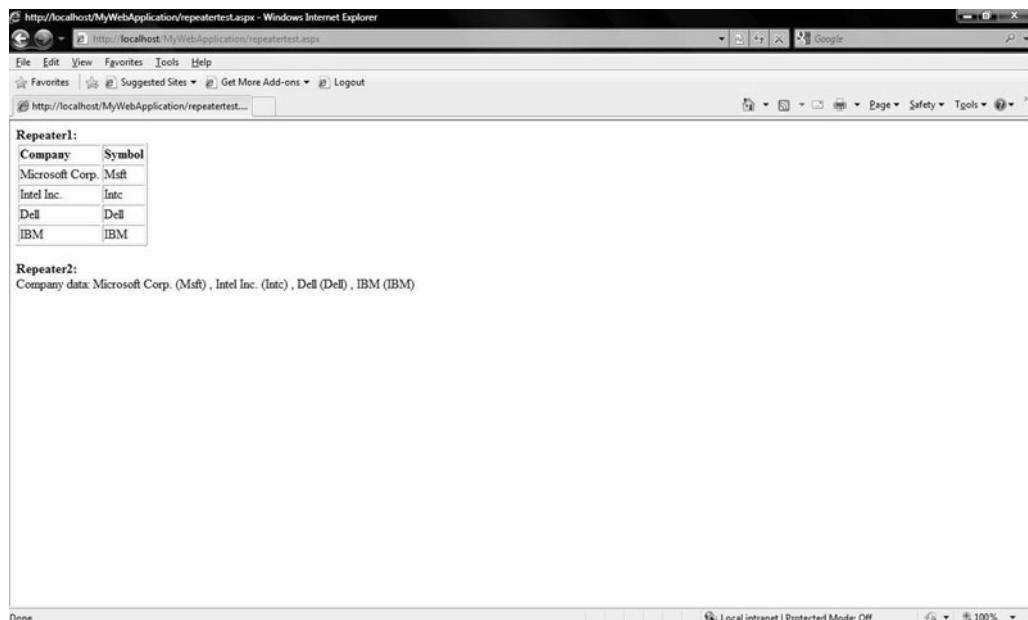
    public PositionData(string name, string ticker)
    {
        this.companyname = name;
        this.ticker = ticker;
    }
    public string Name
    {
        get
        {
            return companyname;
        }
    }
    public string Ticker
    {
        get
        {
            return ticker;
        }
    }
}
</script>

```

Figure 2-8 shows the output of the Repeater control example.

Figure 2-8

Displaying data in the Repeater control



Using the DataPager Control

The DataPager control provides paging functionality for the data-bound controls that implement the `IPageableItemContainer` interface, such as the ListView control.

The **DataPager** class is used to page and display data in navigation controls. You can make use of the **PagedControlID** property to bind data to the **DataPager** class. Alternatively, you can put the **DataPager** control inside the data-bound control, for example, in the **ListView** control, inside the **LayoutTemplate**. You can customize the number of items that are displayed for each page of data by changing the **PageSize** property.

If you want to enable navigation using the **DataPager** control, you must add pager fields to the controls. These pager fields are derived from the **DataPagerField** class. There are three types of pager fields:

- **NextPreviousPagerField**: Enables navigation page by page, or by jumping to the first or last page.
- **NumericPagerField**: Allows navigation by page numbers.
- **TemplatePagerField**: Allows you to create a custom paging UI.

Some important read-only properties of the **DataPager** control are:

- **MaximumRows**: Returns the maximum number of records displayed for each page of data.
- **StartRowIndex**: Returns the index of the first record that is displayed on a page of data.
- **TotalRowCount**: Returns the total number of records that are available in the underlying data source.

The **DataPager** control class is derived from the **Control** class and implements the **IAttributeAccessor**, **INamingContainer**, and **ICompositeControlDesigner** interfaces.

Suppose you want to display the product names and their price page by page using the **ListView** and **DataPager** controls. You also want to provide buttons for navigation to the first and last page.

First add the **ListView** control to the page as shown:

```
<asp:ListView ID="ProductsListView" GroupItemCount="1" runat="server">
    <LayoutTemplate>
        <table cellpadding="2" width="640px" id="tbl1" runat="server">
            <tr>
                <th colspan="5">Products List</th>
            </tr>
            <tr runat="server" id="groupPlaceholder"></tr>
        </table>
    </LayoutTemplate>
    <GroupTemplate>
        <tr runat="server" id="tr1">
            <td runat="server" id="itemPlaceholder"></td>
        </tr>
    </GroupTemplate>
    <GroupSeparatorTemplate>
        <tr id="Tr1" runat="server">
            <td colspan="5">
                <div class="groupSeparator"><hr></div>
            </td>
        </tr>
    </GroupSeparatorTemplate>
    <ItemTemplate>
        <td id="Td1" align="center" runat="server">
            Product Name: <b> <%#Eval("Name") %></b>
            Price:<b> <%# Eval("Price", "{0:c}")%></b> <br />
        </td>
    </ItemTemplate>
```

```

<ItemSeparatorTemplate>
    <td id="Td2" class="itemSeparator" runat="server">&nbsp;</td>
</ItemSeparatorTemplate>
</asp:ListView>

```

Next, for data binding, add this code in the .aspx page before the <HTML> tag. This code will bind the collection of the product names to the ListView control:

```

<script language="C#" runat="server">
    void Page_Load(Object Sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            ArrayList values = new ArrayList();
            values.Add(new ProductData("Nokia N73", "13000.00"));
            values.Add(new ProductData("MotoMing", "9870.00"));
            values.Add(new ProductData("Sony Cybershots", "12000.00"));
            values.Add(new ProductData("MotoRazor", "8000.00"));

            ProductsListView.DataSource = values;
            ProductsListView.DataBind();
        }
    }
    public class ProductData
    {
        private string productName;
        private string price;

        public ProductData(string name, string price)
        {
            this.productName = name;
            this.price = price;
        }

        public string Name
        {
            get
            {
                return productName;
            }
        }

        public string Price
        {
            get
            {
                return price;
            }
        }
    }
</script>

```

Now that you have added the ListView control and its data-binding logic, the next step is to add the DataPager control for pagination:

```

<asp:DataPager runat="server" ID="BeforeListDataPager"
PagedControlID="ProductsListView" PageSize="1">
    <Fields>
        <asp:NextPreviousPagerField ButtonType="Button"
ShowFirstPageButton="true" ShowNextPageButton="false" ShowPreviousPage
Button="false" />

```

CERTIFICATION READY?

Bind controls to data by using data-binding syntax.

3.5

CERTIFICATION READY?

Implement data-bound controls.

2.1

```
<asp:NumericPagerField ButtonCount="10" />
<asp:NextPreviousPagerFieldButtonType="Button" ShowLastPageButton="true" ShowNextPageButton="false" ShowPreviousPageButton="false" />
</Fields>
</asp:DataPager>
```

Ensure that you set the value of the `PagedControlID` property to `ProductListView`, which is the ID of the `ListView` control that you created earlier.

SKILL SUMMARY

In this lesson, you learned about the stages of the life cycle of the ASP.NET page, the events raised during the life cycle, and the difference between HTML server controls and web server controls and when to use each. You also learned about the view state and its use in ASP.NET web application development.

The standard web server controls include the `TextBox`, `Label`, `Button`, `CheckBox`, and `RadioButton` controls. You learned about the features of these controls; their important properties; and events that are useful while developing web applications, such as the `Click` event of a `Button` control, the `Text` property for the `Label` and `TextBox` controls, and the `EnableViewState` property. You also learned about the security measures that you needed to take when setting the `Text` property for these controls.

In addition, you learned about specialized web server controls, such as `Table`, `TableRow`, and `TableCell`, which are useful for displaying data in a tabular fashion. The `Calendar` control is used as a date picker, and the `ImageMap` and `ImageButton` controls are used to display images and have an action associated when the user clicks on them. You also learned how to use the `HotSpot` control. The `Panel` control is a container control that can have multiple child controls.

The `FileUpload` control is used to upload files from the client machine to the server directory. You learned about the different ways of reading a file at the server-side and saving it to the disk. You also learned about security threats for the `FileUpload` control. Next, you learned about the `Wizard` control, which is used to display a series of steps called `WizardSteps`, and its navigation between steps.

`DataBound` controls are used to display data from a data source, such as XML files or a database, to the user. You learned how the `DataSource` object is used. You also learned about the `DataBinder` class and its methods.

You also learned that the `Template` control does not have a UI, and it provides a mechanism for binding to data. You learned about different templates and their usage.

You learned that `ListControl` is an abstract class for common features. You also learned about the controls derived from `ListControl`, such as `DropDownList`, `ListBox`, `CheckBoxList`, `RadioButtonList`, and `BulletedList`.

You learned about the `CompositeDataBoundControl` class that implements the `INamingContainer` interface and about its child controls: `FormView`, `DetailsView`, and `GridView`. The `FormView` control is used to display a single record in a user-defined way. The `DetailsView` is a tabular format of data where each row represents a field in a record. The `GridView` is used to show data in tabular format.

You also learned that the `TreeView` and `Menu` controls are used to show data in a hierarchical fashion. These controls are derived from the `HierarchicalDataBoundControl` class. You also learned about the `DataGrid` and `DataList` controls. `DataGrid` allows you to display data in a table. It also allows you to sort, add, edit, and delete data along with paging. `DataList` helps you display data by applying templates to it. You also learned about the `Repeater` control, which is used to display information by applying custom templates for each item. The `DataPager` control provides the paging functionality for data-bound controls. The `PagedControlID` property of this control is used to bind the data to the `DataPager` control.

(continued)

SKILL SUMMARY (*continued*)

For the certification examination:

- Understand page lifecycle, page flow, and server controls
- Handle page, control, application, and session events.
- Consume standard controls, such as TextBox, Label, Button, CheckBox, FileUpload, and HotSpot.
- Bind controls to data by using data-binding syntax.
- Create collections and implement data-bound controls, such as GridView and DetailsView.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. Web server controls fall under a hierarchy of classes and are derived from the _____ class.
2. The Label control is derived from _____ interface.
3. The Image class contains two subclasses, ImageMap and _____.
4. There are three categories of data-bound controls: simple, composite, and _____.

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. Which control is derived from the IEditableTextControl interface?
 - a. Label
 - b. Button
 - c. TextBox
 - d. CheckBox
2. From which of the following classes is the DataBinder class derived?
 - a. Object
 - b. WebControl
 - c. Control
 - d. Page
3. Which of the following is an abstract class?
 - a. TextBox
 - b. ListControl
 - c. Button
 - d. DropDownList
4. Which of the following controls is a hierarchical data-bound control?
 - a. TableRow
 - b. ImageMap
 - c. Calendar
 - d. TreeView

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---|---|
| T | F 1. View state is used to store data between different postbacks. |
| T | F 2. The RadioButton control is derived from the CheckBox control. |
| T | F 3. A Table control can contain GridViewRow objects. |
| T | F 4. The Panel control is a container control. |
| T | F 5. The GridView control is a hierarchical data-bound control. |

Review Questions

1. What is the importance of the `Application` class?
2. Which event is fired when a Button control is clicked?
3. Can you add a Panel control inside another Panel?
4. Which security measures should you consider when using AdRotator controls?
5. What is the responsibility of the `DataBinder` class?
6. How will you choose between HTML and web server controls?
7. What is the purpose of a Wizard control?
8. What is the use of the FormView control?
9. Explain the various types of nodes in TreeView controls?
10. How should you use the ImageButton control?

■ Case Scenarios

Scenario 1-1: Using Controls in a Web Site

You are creating the Web site for the IT Recruitment Consultancy using ASP.NET 3.5 and Microsoft SQL Server 2008. Job Seekers have to register themselves on this Web site and provide details of their technical skills. You need to save all these details in the database. Which controls will you use to design your site?

Scenario 1-2: Using Controls to Display Reports

You are developing an MIS reporting system for a financial firm using ASP.NET 3.5. The system will display various reports to the management in various formats. Which types of controls will be used in developing the system?



Workplace Ready

The Importance of Controls

In web application development, selecting appropriate server controls for your web page is an important factor. Consider that you are developing a web application for a university. The application must include web pages that register students, display generic course information, and display information about courses available for specific periods, along with details about the faculty. How will you decide which server controls are most appropriate for the application?

Working with User Controls and Web Parts

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Working with user controls.	Load user controls dynamically.	2.2
Working with custom web controls.	Create and consume custom controls.	2.3
Using Web Parts controls.	Customize the layout and appearance of a web page.	7.1

KEY TERMS

composite control
custom control

templated user control
user control

Web Parts
zones

When developing web applications, there will come a time when you will want to develop your own controls that are better suited to the business requirements and that save you development time. Consider different users of a system who expect different representation for the business data. For example, senior management might prefer its data in a graphical format instead of a tabular format because it helps them analyze the data and make better decisions. In contrast, team leaders might prefer to view data in a tabular format to get a detailed picture. ASP.NET provides many user controls and custom controls to achieve such goals. You can also combine functionalities of different server controls and provide new functionality.

In this lesson, you will learn how to create user controls, custom controls, and Web Parts controls.

■ Introducing User Controls

In the previous lesson, you learned about the built-in server controls that ASP.NET provides, including basic, specialized, and data-bound server controls. Apart from these predefined controls, you can create your own controls to suit your requirements. Suppose you want to develop a web application in which you provide administrators and regular users with different user interface (UI) screens for logging on. Though the appearance of the screens might differ, the basic authentication logic remains the same. For example, the application will ask users to provide their login credentials and, after authentication, it will display a welcome screen. In this scenario, it is best to create a login control with standard login authentication logic that can be reused for all login UIs. When you create such a user control, the appearance and behavior of all the UIs based on it will be uniform and consistent. To modify the behavior and/or appearance of the UIs, you only need to modify the user control.



THE BOTTOM LINE

In this section, you will learn what user controls are, how to create basic and templated user controls, and how to use them in a web application.

Utilizing User Controls

A **user control** is a developer-created control that provides application specific behavior and a graphical user interface (GUI). To utilize user controls in your web application, you first need to add existing web server controls to the user control and then define methods and properties according to the application requirements. You can also add an already created user control into a new user control to be utilized on a web page.

A user control is similar to an ASP.NET web page that contains a UI and its corresponding source code. You can create a user control in the same way as a web page, with the markup and the child controls. The code that you add to the user control is for logical implementation of the functionality or data binding, similar to that of a web page. However, there are some differences between user controls and web pages as well. User controls:

- Are saved with the .ascx file extension, whereas web pages are saved as .aspx files.
- Have the @Control directive, whereas web pages have the @Page directive.
- Cannot contain HTML, BODY, or FORM tags, but web pages can.
- Cannot run as standalone entities, but web pages can. User controls need to be included in a web page to expose their functionality.

You can use all the HTML tags (other than HTML, BODY, FORM, and HEAD) and web server controls such as TextBox and Button in a user control. For example, if you want to create a user control to be used as a toolbar, you can add a series of Button controls and handle the Click events on those buttons.

User controls are derived from the `UserControl` class, which in turn is derived from the `TemplateControl` class. The `TemplateControl` class is derived from the `Control` class.

CREATING AND ADDING A USER CONTROL TO A WEB PAGE

In the previous section of this lesson, you learned the basics of user controls. In this section, you will learn to create a user control and add it to a web page.



CREATE A USER CONTROL

Suppose you want to create a user control that receives employee identification number in a text box, and when the employees click the Process Data button, information about their monthly income tax or payroll is processed and displayed.

To create the user control:

1. Right click the project, and select Add New Item.
2. In the New Item dialog box, select Web User Control, and name it `MyWebUserControl`.
3. When you open the newly added user control in the Code view, you can see the `@Control` directive as shown in the following code:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="MyWebUserControl.ascx.cs"
Inherits="UserControlSample.MyWebUserControl" %>
```

TAKE NOTE *

In Solution Explorer, the icon for the user control looks like a web page.

In this snippet, the `@Control` directive defines the user control that is used by the ASP.NET compiler to compile the control. The `Inherits` attribute is the code-behind class for the user control. This attribute can be set to any class that is derived from the `UserControl` class.

You can also specify the `CodeFile` attribute. The value of this attribute specifies the name of the code-behind file for the user control. The `CodeFile` attribute is used along with the `Inherits` attribute to associate the code-behind file with the user control.

4. To the newly created control, MyWebUserControl, add a Label, TextBox, and Button control as shown in this code example:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="MyWebUserControl.ascx.cs"
Inherits="UserControlSample.MyWebUserControl" %>
<asp:Label ID="myLabel" Text="Enter Data"
runat="server"/> &nbsp;
<asp:TextBox ID="txtData" runat="server"/>
<br /> <br />
<asp:Button ID="btnProcessData" runat="server"
Text = "Process Data" OnClick ="btnProcessData_Click"/>
```

Note that the `OnClick` method raises the `Click` event of the Button control. Raising the `Click` event invokes the click event handler (`btnProcessData_Click`).

5. After you have created the user control, you should add code for the `OnClick` event handler in `MyWebUserControl.ascx.cs`. This file becomes the code-behind file for the user control.
6. To use the created user control, you need to add it to a web page. The following code example shows the code for the `MyWebForm` page, which hosts the `MyWebUserControl` control:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="MyWebForm.aspx.cs" Inherits="UserControlSample.
MyWebForm" %>
<%@ Register Src="~/MyWebUserControl.ascx" TagName="MyControl"
TagPrefix="uc1"%>
<!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></title>
</head>
<body>
<form id="form1" runat="server">
<div>
<uc1:MyControl ID="MyControl1" runat="server" />
</div>
</form>
</body>
</html>
```

In this code snippet, the value of the `Src` attribute is set to `"~/MyWebUserControl.ascx"` and the tilde character indicates the root directory of the application.

Note below the `@Page` directive you'll find the `@Register` directive, which is used to register the user control.

The `@Register` directive contains the following attributes:

- **TagPrefix:** Is the namespace identifier for the user control. Visual Studio sets the default value for this attribute to `uc1` (denoting User Control 1, the first user control on the web page).
- **TagName:** Indicates the name for the control.
- **Src:** Specifies the location or path of the user control.

On the web page, the instance of the user control is created in the `<form>` tag. You can also add more than one instance of the user control on your web page.

You can add the @Register directive in the Web.config file so that the user control gets registered in all the web pages of your application. You should add this under the Pages→Controls section of System.Web.

TAKE NOTE *

The @Reference directive is also used to indicate that the user control should be dynamically linked in the current web page in which this @Reference directive is mentioned.

You can use the following syntax to do this:

```
%@Reference Page="~/MyWebForm.aspx" Control("~/MyWebUserControl.ascx")%
```

Using the @Reference directive dynamically compiles the user control page and links it to the current page. Therefore, you can refer to the external compiled objects from the current file.

If you have already created a web page that you want to reuse in many locations, you can convert the web page itself into a user control. To do this:

TAKE NOTE *

1. Remove the HTML, BODY, and FORM tags.
2. Change the @Page directive at the top of the page to the @Control directive.
3. In the @Control directive, change the value of the Inherits attribute from System.Web.UI.Page type to System.Web.UI.UserControl type.
4. Change the file extension from .aspx to .ascx.

ACCESSING USER CONTROL DATA

Now that you have added a user control to a web page, the Label, TextBox, and Button controls are visible to the user. All the controls added in the MyWebUserControl user control are protected members. You can quickly navigate to the designer-generated code file to verify that the access modifiers are protected. This means that these controls are accessible only to the MyWebUserControl class and its derived classes. You can access only those properties that are exposed by the control. To access these controls from the web page, you need to expose their properties in the code behind of your user control as shown here:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace MyNamespace
{
    public partial class MyWebUserControl : System.Web.UI.UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        public string Caption
        {
            get { return this.myLabel.Text; }
            set { this.myLabel.Text = value; }
        }

        public string UserText
        {
            get { return this.txtData.Text; }
            set { this.txtData.Text = value; }
        }
    }
}
```

TAKE NOTE *

If you are setting properties that are not tied to a control that has ViewState enabled and you want them to persist across postbacks, you need to save the values in the ViewState rather than as instance variables.

X REF

To know more about layouts, refer to Lesson 1 “Introducing ASP.NET 3.5.”

```
protected void btnProcessData_Click(object sender, EventArgs e)
{
    }
}
```

Now assign values for the Label and TextBox properties by accessing them in the web page as shown:

```
protected void Page_Load(object sender, EventArgs e)
{
    this.MyControl1.Caption = "Enter your ID: ";
    this.MyControl1.UserText = "Enter data here";
}
```

POSITIONING USER CONTROLS

Arranging controls on a web page is called layout management. ASP.NET provides flow layout and grid layout to arrange your controls. Flow layout arranges the controls in left to right and top to bottom order. Grid layout arranges the controls in a grid.

You can position server controls on a web page using grid layout. For example, you can place the controls in an HTML table. However, you cannot set the absolute position of a user control using the **Style** property because the user control does not add an outer tag for the contents of the control, which could be assigned a style. Therefore, you need to use a **Panel** control on the web page and place the user control in the panel, and then position the panel and its contents.

DYNAMICALLY LOADING CONTROLS

You now know how to create a user control and add it to a web page. You used a static method to do this. ASP.NET also allows you to load user controls dynamically. This is extremely useful when you want to display multiple user controls on your web page. In the following code example, two instances of the **MyWebUserControl** are added to the web page dynamically:

```
protected void Page_Load(object sender, EventArgs e)
{
    MyWebUserControl control1 = (MyWebUserControl)
LoadControl("MyWebUserControl.ascx");
    control1.Caption = "Enter Data for Control 1";
    Form.Controls.Add(control1);

    MyWebUserControl control2 =
(MyWebUserControl)LoadControl("MyWebUserControl.ascx");
    control2.Caption = "Enter Data for Control 2";
    Form.Controls.Add(control2);
}
```

The **LoadControl** method loads the control into memory by accepting the path of the .ascx file and returning an object of the **System.Web.UI.Control** class. You can access the properties of the user control by type casting it to an object of the **MyWebUserControl** class. Once the user control is loaded into memory, you can add it to the form using the **Form.Controls.Add** method so that it is visible on your web page.

RAISING EVENTS TO THE PAGE

Events are notification messages that indicate the occurrence of some action. Consider a situation where you are developing a user control that asks the user to input some data. The control also has a Button control that further processes the data entered by the user. However, when designing the control, if you are not aware of what logic must be added in the button's

Click event, you can fire an event to the hosting web page. This event handler will implement the necessary steps.

User controls can have their own events that can post the form data of the web page back to the server. Because there can be only one form server control in a web page, user controls do not contain form server data. User controls are programmed to be aware of the web page life cycle and have many of the same events that the web page has, such as the Load and Init events. In the MyWebUserControl example, you can write your code in the OnClick event handler for the Process Data button.

TAKE NOTE*

Typically, developers expose the Button control in the user control by making it public and then attempting to attach the event handler to it. However, this is not a good practice. A better way would be to declare a delegate public rather than making the Button control public.

A delegate represents a pointer or reference to the method.

This next code example shows the code-behind file in which a public delegate has been added at the top of the class. The Click event's handler method for the ProcessData button will raise this event by passing the value entered in the text box:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public delegate void ProcessDataHandler(string message);

namespace UserControlSample
{
    public partial class ProcessDataControl: System.Web.UI.UserControl
    {
        public event ProcessDataHandler ProcessData;
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void btnProcessData_Click(object sender, EventArgs e)
        {
            if (ProcessData != null) ProcessData(txtEnterData.Text);
        }
    }
}
```

You add this user control, say ProcessDataPage.aspx, to the web page as follows:

```
<%@ Page Language="C#"
AutoEventWireup="true" CodeBehind="ProcessDataPage.aspx.cs"
Inherits="UserControlSample.ProcessDataPage" %>

<%@ Register Src "~/ProcessDataControl.ascx" TagName="ProcessData"
TagPrefix="uc1" %>
<!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></title>
    </head>
```

```

<body>
    <form id="form1" runat="server">
        <div>
            <uc1:ProcessData runat="server" ID="UserControl1"/>
        </div>
    </form>
</body>
</html>

```

In the code-behind file, you need to register the event handler for `UserControl1.ProcessData`. In the `UserControl1_ProcessData` method, display the message to the user about the received data as follows:

```

using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

namespace UserControlSample
{
    public partial class ProcessDataPage : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            UserControl1.ProcessData 1= new ProcessDataHandler
(UserControl1_ProcessData);
        }
        void UserControl1_ProcessData(string message)
        {
            Response.Write("Data Received from User Control
is " + message);
        }
    }
}

```

In the previous code, the `Page_Load` method registers the handler method for the `ProcessData` event using the delegate. Whenever the user control raises the `ProcessData` event, the web page is notified and invokes the `UserControl1_ProcessData` method.

CREATING A TEMPLATED USER CONTROL

When developing web applications, separation of control data from its presentation is an important aspect because it makes your application flexible and loosely coupled. For example, you can separate the business logic from the presentation in an application. As a result, you would be able to make changes to the presentation or the business strategy independently. In addition, a flexible and loosely coupled application allows designers and developers to work together without conflicting each other's work. You can achieve this separation with the help of ***templated user controls***. These controls do not provide any default UI. For example, if you need to display customer information, such as a customer's ID, name, and phone number, and you do not know how the web designer will format this information, you can create a templated user control called `CustomerControl`. This control will allow the page designer to supply the format for customer data using a template.

Similar to user controls, templated user controls are only reusable in the same Web site. They must provide a container class (a naming container) that has properties that are accessible to the hosting web page and provide the unique namespace ID. The template contains the UI for the templated user control supplied by the page developer during design time. These templates can contain markup and controls.



CREATE A TEMPLATED USER CONTROL

To create a templated user control:

1. Add a user control file in your web application.
2. In the .ascx file, add an ASP.NET PlaceHolder control where you want the templated user control to appear:
`<asp:PlaceHolder ID="CustomerPlaceHolder" runat="server" />`
3. In the code-behind file of the user control, implement a property of type ITemplate:

```
[TemplateContainer(typeof(CustomerContainer))]  
public ITemplate CustomerTemplate  
{ get; set; }
```

4. Add a new class to the App_Code folder in the web application that contains the naming container class for the templated control. This class must inherit the Control class and implement the INamingContainer interface. Additionally, the naming container class should provide the public properties for each data element that is visible to the template. When the container control gets rendered, it contains an instance of the template. The source code in the App_Code folder is compiled at runtime, and the compilation result is available to the web application.
5. Apply the TemplateContainerAttribute to the ITemplate property and then pass the type of the template's naming container class as an argument to the constructor of the attribute.
6. Apply the PersistenceModeAttribute to the ITemplate property and pass the template's naming container class as an argument to the constructor of the attribute. This attribute specifies metadata that defines how the properties or events are persisted on an ASP.NET page at design time:

```
[PersistenceMode(PersistenceMode.InnerProperty)]  
[TemplateContainer(typeof(CustomerContainer))]  
public ITemplate CustomerTemplate;
```

7. In the source code of the user control page, add public properties to pass your data to the template naming container class so that the data is available in the template:

```
public int CustomerID  
{ get; set; }  
public string CustomerName  
{ get; set; }  
public string CustomerPhone  
{ get; set; }
```

8. In the Page_Init method of the user control, test for the ITemplate property being set. If the property is set to null, create an instance of the naming container class, and create an instance of template in the naming container. Add the instance of the naming container class to the Controls property of the PlaceHolder server control:

```
public void Page_Init()  
{  
    CustomerPlaceHolder.Controls.Clear();  
    if (CustomerTemplate == null)  
    {
```

```

        CustomerPlaceHolder.Controls.Add(new
LiteralControl("Template not defined."));
        return;
    }
    CustomerContainer c = new CustomerContainer();
    c.CustomerID = this.CustomerID;
    c.CustomerName = this.CustomerName;
    this.CustomerPhone = this.CustomerPhone;
    CustomerTemplate.InstantiateIn(c);
    CustomerPlaceHolder.Controls.Add(c);
}

```

The following code sample defines a templated user control called CustomerControl.ascx. This templated user control allows you to set the CustomerID, CustomerName, and CustomerPhone, and allows you to create a template to define the output. Here is the C# code for the .ascx file:

```

<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="CustomerControl.ascx.cs"
Inherits="TemplatedUserControl.CustomerControl" %>
<asp:PlaceHolder ID="CustomerPlaceHolder" runat="server" />

```

Now, let's have a look at your code-behind file:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace TemplatedUserControl
{
    public partial class CustomerControl : System.Web.UI.UserControl
    {
        [PersistenceMode(PersistenceMode.InnerProperty)]
        [TemplateContainer(typeof(CustomerContainer))]
        public ITemplate CustomerTemplate
        {
            get; set;
        }
        public int CustomerID
        {
            get; set;
        }
        public string CustomerName
        {
            get; set;
        }
        public string CustomerPhone
        {
            get; set;
        }
        public void Page_Init()
        {
            CustomerPlaceHolder.Controls.Clear();
            if (CustomerTemplate == null)
            {
                CustomerPlaceHolder.Controls.Add(new
LiteralControl("Template not defined."));
                return;
            }
            CustomerContainer c = new CustomerContainer();
            c.CustomerID = this.CustomerID;
            c.CustomerName = this.CustomerName;
            this.CustomerPhone = this.CustomerPhone;
            CustomerTemplate.InstantiateIn(c);
        }
    }
}

```

```
        CustomerPlaceHolder.Controls.Add(c);
    }
protected void Page_Load(object sender, EventArgs e)
{
}
}
Code for the naming container class:
using System;
using System.Web;
using System.Web.UI;
namespace TemplatedUserControl
{
    public class CustomerContainer: Control, INamingContainer
    {
        public CustomerContainer()
        {
            public int CustomerID { get; set; }
            public string CustomerName { get; set; }
            public string CustomerPhone { get; set; }
        }
    }
}
```

The templated user control is created, as shown in Figure 3-1.

Figure 3-1
Templated user control



USING THE TEMPLATED USER CONTROL

Templated user controls must be used within the same project. To use a templated control, you first need to add the control to the web page, set its properties, and then, add the template. The following code example shows a sample web page that contains the templated control, CustomerControl, with a template to format customer data:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="CustomerPage.aspx.cs"
Inherits="TemplatedUserControl.CustomerPage" %>
```

```

<%@ Register Src="~/CustomerControl.ascx" TagName="CustomerControl"
TagPrefix="uc1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    public void Page_Load()
    {
        DataBind();
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title></title>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <uc1:CustomerControl ID="CustomerControl1" runat="server"
CustomerID="1" CustomerName="Lamar" CustomerPhone="(501) 444- 9843">
                    <CustomerTemplate>
                        <h1>Customer Information</h1>
                        <span style="background-color:Lime">&nbsp;ID:&nbsp;</span>
                        <%#Container.CustomerID %>
                        <span style="background-color:Lime">&nbsp;Name:&nbsp;</span>
                        <%# Container.CustomerName %>
                        <span style="background-color:Lime">&nbsp;Phone:&nbsp;</span>
                        <%# Container.CustomerPhone %>
                    </CustomerTemplate>
                </uc1:CustomerControl>
            </div>
        </form>
    </body>
</html>

```

CERTIFICATION READY?
Load user controls
dynamically.
2.2

TAKE NOTE *

You cannot view the interface of a templated user control during design time. Therefore, make sure that you run your web page to verify that it displays properly.

■ Working with Custom Web Controls



Suppose you are creating a Web site for a department store that allows users to browse available products and add them to a shopping cart. The selected products can later be bought or saved to the user's wish list. The Web site also facilitates online payment using credit cards. The data that it accepts as input includes credit card number, expiration date, and CVV code, which when authenticated completes the transaction successfully. Input fields related to credit card details, such as the card number and the CVV code fields, should not accept alphabets and special characters. Using standard Web server controls to validate these fields causes redundancy of the validation logic because you need to write the same code in every control. Instead, you can create a custom control with all the necessary validation logic and reuse this control across multiple pages in your Web site.

In this section, you will learn what custom controls are, how to create them, and how to use these controls in a web application.

Creating a Custom Web Server Control

A **custom control** is a web server control that is derived from the `WebControl` or `Control` class. You can use the `WebControl` class to add properties, such as `BackColor`, `Font`, etc. You can derive your custom control from the `Control` class and implement such properties.

While creating a custom control, you can make use of the built-in server controls. This approach provides more reusability and saves development time. The `WebControl` class provides the `Style` property, which includes UI-related attributes such as `ForeColor`, `BackColor`, `Font`, `Height`, and `Width`.

Based on the amount of reusability of the control, you can create a custom web control by deriving it in two different ways:

- From another custom control, or
- Directly from the `WebControl` class or any of its derived classes

Suppose you need to create a control based on a control called `ProcessData`. You also want to use the new control to process customer payment data. In this case, you can reuse the `ProcessData` control and just change the business logic. However, if you need a control that shows graphs and charts, you may need to create a new control.

If you want to create several custom controls and share them among multiple web applications, you can create them as a class library of custom controls that form an assembly, that is, a `.dll`. To consume a custom control from an assembly in your application, simply add the reference to the `.dll` file.

CREATING A CUSTOM CONTROL USING ANOTHER CONTROL

When you create a custom control using an existing control, you can add more methods, properties, and events. You can also override existing properties and methods to obtain different behaviors. For example, if you want to create a text box with a label that exposes the `LabelCaption` property in the `span` tag, you need to inherit your control from the `TextBox` control and add new properties, as shown in this code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI.WebControls;

namespace CustomControlSample
{
    public class MyTextBox: TextBox
    {
        public int LabelWidth { get; set; }
        public string LabelCaption { get; set; }
        protected override void Render(System.Web.UI.HtmlTextWriter
writer)
        {
            writer.Write(@"<span
style=""display:inline-block;width:{0}px"">{1}&nbsp;</span>",
LabelWidth, LabelCaption);
            base.Render(writer);
        }
    }
}
```

Note that the `Render` method has been overridden to add the new features.

Next, to use the custom control, you need to add it in your web page statically or dynamically.



ADD A CUSTOM CONTROL TO A WEB PAGE

To add a custom control to a web page:

1. Open the web page in the designer mode.
2. From the ToolBox, expand CustomerControlSample Components. This is the namespace of your custom control or you may see the name of the web application project.
3. Select MyTextBox, and drag and drop this control on the web page. Then you can use the custom control exactly the same way you use standard controls such as Button or TextBox.

You can also add the custom control dynamically on your web page. To add the MyTextBox custom control dynamically on a web page, use the following code:

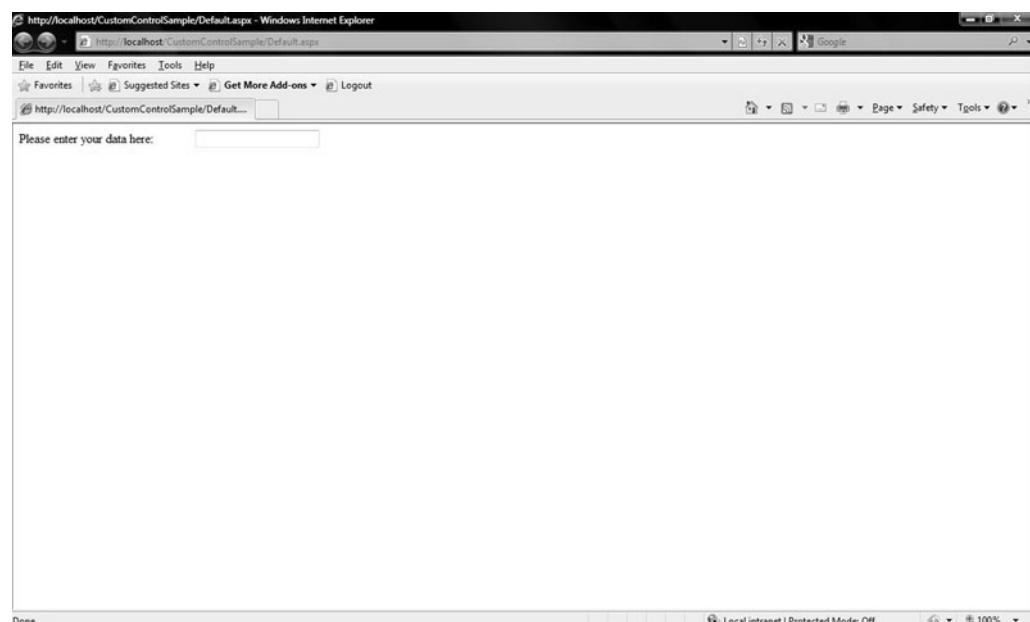
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControlSample
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            MyTextBox t = new MyTextBox();
            t.LabelCaption = "Please enter your data here: ";
            t.LabelWidth = 220;
            Form.Controls.Add(t);
        }
    }
}
```

The custom control is added to the web page as shown in Figure 3-2.

Figure 3-2

Control rendered in browser



CREATING A CUSTOM CONTROL FROM THE WEBCONTROL CLASS

In addition to inheriting from built-in Web server controls, you can create your custom controls by inheriting them from the `WebControl` class. This is usually done when you do not find a built-in control with the base behavior that you want. The following code example shows the creation of a custom control (derived from the `WebControl` class) that displays images along with the image names:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI.WebControls;

namespace CustomControlSample
{
    public class ImageControl: WebControl
    {
        public string ImageCaption
        {
            get;
            set;
        }

        public string ImagePath
        {
            get;
            set;
        }

        protected override void Render
        (System.Web.UI.HtmlTextWriter writer)
        {
            writer.WriteFullBeginTag("div");
            writer.Write(@"<img src=""{0}""/> <br/>", ImagePath);
            writer.Write(ImageCaption + "<br/>");
            writer.WriteEndTag("div");
        }
    }
}
```

When the control is rendered, a `<div>` tag is sent to the browser. The `<div>` tag contains an `` tag whose source is set to `ImagePath` and whose caption is set to `ImageCaption`.

To test the created control, add a new web form, `ImageControlTest`, to the web application. Then add the following code in the `Page_Load` method:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CustomControlSample
{
    public partial class ImageControlTest: System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
```

```

        ImageControl ic = new ImageControl();
        ic.ImageCaption = "Test Image";
        icImagePath = @"test.jpg";
        Form.Controls.Add(ic);
    }
}
}

```

TAKE NOTE *

When you derive a control from the `WebControl` class and render a single element, consider using the `RenderContents` method instead of the `Render` method because the `Render` method calls the `RenderContent` method after rendering the opening tags and its style attributes. When you override the `Render` method for writing contents, your control will lose the style rendering logic.

ADDING A CUSTOM WEB SERVER CONTROL TO THE TOOLBOX

In the examples you have seen so far, the custom controls created were added statically or dynamically to a web page. You can also add these controls to the Web Designer ToolBox so that you can drag and drop them onto your form. As a best practice, you should first create an assembly of custom controls, and then, use it in your web application. To add a control that is present in an assembly, you need to place the custom control in a separate .dll file, for example `MyCustomWebControlLibrary.dll`, which has all the custom controls in your web application. You would then compile the code in the assembly to generate it.



ADD A CUSTOM CONTROL PRESENT IN AN ASSEMBLY

To add a custom control present in an assembly:

1. Right click the ToolBox.
2. Select Choose Items, and browse for the `MyCustomWebControlLibrary.dll` file.

All controls available in `MyCustomWebControlLibrary` will be added to the ToolBox.

When the custom control is added to the ToolBox, it shows the default icon. If you want to change the icon, add an attribute `ToolboxBitmap`, and set its value to the path of an icon bitmap sized at 16 × 16 pixels. Ensure that the path to the icon file is an absolute path. You can also specify a bitmap file as an icon. This will be an embedded resource as long as the name of the file is the same as the fully qualified name of the class (such as `ImageControl.bmp`).

The `ToolboxBitmapAttribute` class is present in the `System.Drawing` namespace. The following code example shows you how to set the `ToolboxBitmap` attribute:

```
[ToolboxBitmap(typeof(ImageControl),
"MyCustomWebControlLibrary.ImageControl.bmp")]
public class ImageControl: WebControl
```

Individualizing a Custom Web Server Control

When you add a custom control to the ToolBox, it displays control-specific generic information, such as version number and a brief explanation as a tooltip.

If you want to change the information that your custom Web server control displays, you can use the `ToolboxDataAttribute` class for the control class. The `ToolboxDataAttribute` default tag is generated for a custom control when it is dragged from the ToolBox:

```
[ToolboxBitmap(typeof(ImageControl), "MyCustomWebControlLibrary.
ImageControl.bmp")]
[ToolboxData(@"<{0}: ImageControl runat=""server"""
CompanyName="" ImagePath="" ""/>")]
public class ImageControl: WebControl
```

When you drag a control to a web page, it will display the default namespace. If you want to change the default namespace to a specific one, you can use the `TagPrefixAttribute` class. You should apply this attribute to the custom control class. You can also specify a namespace prefix that is assigned by the web page designer to the assembly that contains the custom control using the following code:

TAKE NOTE *

Note that using a designer is optional; your control will run without one.

```
[assembly: TagPrefix("MyCustomWebControlLibrary", "mcwl")]
```

CREATING A CUSTOM DESIGNER FOR A CUSTOM CONTROL

Custom designers are used to render custom controls in the design mode. To apply a custom designer to your custom control, you need to add a `System.Design.dll` reference and create a class that inherits the `ControlDesigner` class.

Consider using a custom designer when the normal rendering of the control is not visible, which is usually caused by the code that runs to populate some properties of the control.

For example, to provide a custom designer for the custom control `ImageControl` that provides a default rendering when the `ImagePath` is not set, use the following code:

```
using System;
using System.Collections.Generic;
using System.Web;
using System.Web.UI.Design;

namespace CustomControlSample
{
    public class ImageControlDesigner : System.Web.UI.Design.
ControlDesigner
    {
        private ImageControl _imageControl;
        public override string GetDesignTimeHtml()
        {
            if (_imageControl.ImagePath.Trim().Length== 0)
            {
                return "<div id='mc1' "
                +
                "style='background-color:yellow; border-width:2px;' >"
                +
                "<center>ImageControl</center><br />"
                +
                "<center>Please set ImagePath property.</center><br />"
                + "</div>";
            }
            else
            {
                return base.GetDesignTimeHtml();
            }
        }

        public override void Initialize(IComponent component)
        {
            _imageControl = (ImageControl)component;
            base.Initialize(component);
            return;
        }
    }
}
```

When applying designers, ensure that you have applied the `DesignerAttribute` to the `ImageControl` class as shown in this code snippet:

```
[Designer("CustomControlSample.ImageControl,
ImageControlDesigner")]public class ImageControl: WebControl
```

CREATING A COMPOSITE CONTROL

When a custom web server control contains other controls, it is called a ***composite control***. A composite control is similar to the user control. However, it does not provide the designer interface and the .ascx file. To create a composite control, you create a class that is inherited from the **CompositeControl** class and then add the constituent controls to the **Controls** collection in the class you have created.

Note that a composite control is rendered as a tree of controls, each with its own life cycle and formation of a new application programming interface (API). Every child control handles its postback data and events on its own. This is very useful because you do not have to write any code for PostBack event handling.

To create your own composite control, create a class that is derived from the **CompositeControl** class and override the **CreateChildControls** method. The **CreateChildControls** method is available in the **Control** class that serves as a base class for **CompositeControl** class. This method should contain the code to instantiate the child controls and set their properties. You can use the **Panel** control to assign styles to your composite control.

ReCreateChildControls method of this class recreates the child controls in a control that is derived from **CompositeControl**.

CREATING A TEMPLATED CUSTOM CONTROL

Templated custom controls are used for the separation of control data and its presentation. Similar to templated user controls, these controls do not provide any UI.

CREATE A TEMPLATED CUSTOM CONTROL

To create a templated custom control:

1. Create a **ClassLibrary.dll** project.
2. Add a **System.Web.dll** reference in the **ClassLibrary.dll** created in Step 1.
3. In the project, add a container class with public properties for the data that you want to access with the help of the container object, for example, **VendorInfoControl** as shown:

```
public class VendorInfoControl: Control, INamingContainer
{
    public int VendorID
    { get; set; }
    public string VendorName
    { get; set; }
    public int PurchaseOrderNumber
    { get; set; }
    public int ItemCode
    { get; set; }
}
```

4. Include the **System.Web.UI** namespace in the container class file.
5. Derive the container class from the **System.Web.UI.Control** class and remember to inherit the **INamingContainer** interface.
6. Add a class **VendorInfoControl** to the project for the templated control.
7. Include the namespace **System.Web.UI** in the class that you created in Step 6.
8. Include the source code for the templated control to derive from the **System.Web.UI.Control** class and the **INamingContainer** interface in the templated control class.

MORE INFORMATION

For more information on the methods of the **Control** class, refer to the Control Class section on MSDN.

TAKE NOTE*

If you are creating many composite controls with similar properties and/or methods, consider creating a base class for the composite control with common features.

9. Add an attribute ParseChildren(true) to the class, which indicates to the page parser that the nested content contained within the server control is parsed as a control and that it is used for setting the properties of the templated control as shown in the code snippet:

```
[ParseChildren(true)]
public class VendorInfoControl: Control, INamingContainer
10. Create properties in the templated control class that have the datatype ITemplate and the template defined in the page designer. Each of these properties should have the TemplateContainer attribute set to the datatype of the container. In addition, these properties must have the PersistenceMode attribute set to the PersistenceMode.Inner property, as shown here:
```

```
[PersistenceMode(PersistenceMode.InnerProperty)]
[TemplateContainer(typeof(VendorContainer))]
public ITemplate VendorTemplate
{ get; set; }
```

11. Override the DataBind method to call the EnsureChildControls method:

```
public override void DataBind()
{
    EnsureChildControls();
    base.DataBind();
}
```

12. Override the CreateChildControls method to instantiate the template using the InstantiateIn method of the ITemplate interface as shown in this code snippet:

```
protected override void CreateChildControls()
{
    Controls.Clear();
    if (VendorTemplate != null)
        VendorContainer v = new VendorContainer(VendorID, Vendor-
Name, PurchaseOrderNumber, ItemCode);
        VendorTemplate.InstantiateIn(v);
        Controls.Add(v);
    }
    else
    {
        this.Controls.Add(new LiteralControl("No VendorTemplate
Defined."));
    }
}
```

The following code example shows the creation of a custom control, VendorInfoControl, derived from the Control class and the INamingContainer interface:

```
using System;
using System.Web.UI;
using System.Web;
namespace CustomControlSample
{
    [ParseChildren(true)]
    public class VendorInfoControl: Control, INamingContainer
    {
        public int VendorID
        { get; set; }
        public string VendorName
        { get; set; }
        public int PurchaseOrderNumber
        { get; set; }
```

```

        public int ItemCode
        { get; set; }
    [PersistenceMode (PersistenceMode.InnerProperty)]
    [TemplateContainer(typeof(VendorContainer))]
    public ITemplate VendorTemplate
        { get; set; }
    public override void DataBind()
    {
        EnsureChildControls();
        base.DataBind();
    }
    protected override void CreateChildControls()
    {
        Controls.Clear();
        if (VendorTemplate != null)
        {
            VendorContainer v = new
VendorContainer(VendorID, VendorName, PurchaseOrderNumber, ItemCode);
            VendorTemplate.InstantiateIn(v);
            Controls.Add(v);
        }
        else
        {
            this.Controls.Add(new
LiteralControl("No VendorTemplate Defined."));
        }
    }
}
}

```

13. Now display the vendor information. To do this, create the naming container, VendorNamingContainer, for the VendorInfoControl custom control, as shown in the following code example:

```

using System;
using System.Web.UI;
using System.Web;
namespace CustomControlSample
{
    public class VendorContainer: Control, INamingContainer
    {
        public int VendorID
        { get; set; }
        public string VendorName
        { get; set; }
        public int PurchaseOrderNumber
        { get; set; }
        public int ItemCode
        { get; set; }
        public VendorContainer()
        {
        }
        public VendorContainer(int vendorID, string vendorName, int
PONumber, int itemCode)
        {
            this.VendorID = vendorID;
        }
    }
}

```

```
        this.VendorName = vendorName;
        this.PurchaseOrderNumber = PONumber;
        this.ItemCode = itemCode;
    }
}
}
```

You will now be able to display the vendor information in your application.

USING THE TEMPLATED CUSTOM CONTROL

To use the templated custom control, you need to convert its project into a .dll file.



USE A TEMPLATED CUSTOM CONTROL

To expose the functionality of the templated custom control:

1. Compile the control library where you added the templated control and generate a .dll file.
2. Right click ToolBox, and select Choose Items.
3. Browse to the .dll generated file, and add the custom control to your ToolBox.

The following code example shows you how to create a sample web page, VendorInfoPage.aspx, and add the custom control VendorInfoControl to the web page:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="VendorInfoPage.aspx.cs"
Inherits="CustomControlSample.VendorInfoPage" %>
<%@ Register assembly="CustomControlSample"
namespace="CustomControlSample" tagprefix="cc1" %>
<script runat="server">
    public void Page_Load()
    {
        DataBind();
    }
</script>
<!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></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
<cc1:VendorInfoControl ID="VendorInfoControl1" runat="server"
VendorID="1" VendorName="Vendor 1"
PurchaseOrderNumber="121" ItemCode="22" >
    <VendorTemplate>
        <h1>Vendor ID: <%#Container.VendorID %></h1>
        <table border="1">
            <tr>
                <td>Name:</td>
                <td><%#Container.VendorName%></td>
            </tr>
            <tr>
                <td>Purchase Order No.:</td>
                <td><%#Container.PurchaseOrderNumber %></td>
            </tr>
            <tr>
```

CERTIFICATION READY?

Create and consume
custom controls.

2.3

```

<td>Item Code:</td>
<td align="right"><%#Container.ItemCode %> </td>
</tr>
</table>
</VendorTemplate>
</cc1:VendorInfoControl>
</div>
</form>
</body>
</html>

```

The control will be added to the .dll file. You can now reuse the .dll file.

TAKE NOTE *

Ensure that you are calling the `.DataBind` method so that the vendor data is bound to the custom control, `VendorInfoControl`, when rendering the control.

UNDERSTANDING USER CONTROLS AND CUSTOM CONTROLS

In the previous sections, you learned about user and custom controls and how to create and use them. There are some differences between user controls and custom controls. You may be wondering when you should use a user control and when you should use a custom control; in other words, what are the basic differences between the two? Table 3-1 lists the differences between a user control and a custom control.

Table 3-1

User controls versus custom controls

USER CONTROLS	CUSTOM CONTROLS
Easy to create	Difficult to create
Provides limited support for UI design for developers who use visual design tools	Provides full support for developers who use visual design tools for UI design
Cannot be added to the ToolBox	Can be added to the ToolBox
Best suited for static layout	Best suited for dynamic layout
Presence of a local copy of the code is mandatory when used across applications	Local copies of the code are not mandatory; can be deployed on to the global assembly cache (GAC) for cross-application usage

■ Introducing Web Parts Controls

THE BOTTOM LINE

Suppose you want to develop a social networking Web site where registered users can change the look and feel of the site, customize their home page, or even rearrange the windows or controls that are visible to them. In addition, when the user signs out, you want to save all the customizations to show them again when they log on the next time. In other words, your Web site should provide support for personalization. ASP.NET facilitates developers in achieving this flexibility by using Web Part controls. Web Part controls are widely used in SharePoint sites. Note that in a SharePoint site, you can very easily create Web Parts on your application using the wizard provided with minimal coding.

Using Web Parts

ASP.NET **Web Parts** are a collection of controls that help create Web sites in which users can modify the contents, behavior, and appearance of the web pages directly from the web

browser. The changes made can be updated in the pages rendered to all the users connected to the site or to specific users. For example, consider widgets or a Web site, such as iGoogle. The web pages are divided into sections. Now, you can customize each section and personalize it. For example, you can select the horoscope section so that you can view your horoscope every time you log on. All this is possible using Web Parts.

You will find it beneficial to use web parts controls in your applications. By using Web Parts, you can design your application so that users can personalize the content and layout as they like. Users can add, hide, minimize, or remove the web parts controls from the page, position the controls to different **zones** on web pages, and modify the appearance and properties.

Using Web Parts, you can export and import settings of web parts controls to use them in other pages or sites. Using this feature, you can preserve the appearance, properties, and data within the Web Parts control. You can also connect the different controls on a web page using Web Parts. For example, you can use Web Parts to connect to different Web sites and display relevant content in the form of charts and graphs.

Using Web Parts, you can also manage site-level settings, such as defining the access and setting role-based access.

Web parts controls have three components:

- **Web Parts UI controls:** Are derived from the `Part` class, which defines the UI on the Web Parts page. You can create your own Web Parts controls or make use of the existing ASP.NET server controls to create new Web Parts.

Imagine that you have created a web page on which you need to add Web Parts controls to all the pages. You also want to apply the changes made to the Web Parts controls on all the pages. How will you achieve this?

To use Web Parts on your web page, you need to use a `WebPartManager` control on every Web Parts page. This control helps manage all the Web Parts controls on the page.

To reuse controls on pages, you can create a catalog of Web Parts controls using the `CatalogZone` controls and then select the appropriate control from the catalog to utilize on your web page. You can also edit and personalize the controls using the `EditorZone` and `EditorPart` controls. Some other important controls that you may use are `WebPartZone`, `CatalogPart`, `ConnectionZone`, etc.

- **UI structural controls:** Provides the core structure and the services required by the Web Parts. Examples of UI structural components are the `WebPartManager` control and zones. The `WebPartManager` control is a structural component that is required on every Web Parts page. This control coordinates all the Web Parts controls on the page. Zones provide UI elements, such as header, footer, buttons, etc. and help manage the layouts of pages.
- **Personalization:** Allows you to personalize the layout, behavior, and appearance of the Web Parts control. These setting changes persist for the current as well as future sessions.

DEVELOPING WEB PARTS CONTROLS

Now that you've learned about Web Parts, let's create a simple Web Parts control that displays a message to the user.



CREATE A WEB PARTS CONTROL

To create this basic Web Part:

1. Open an instance of the Visual Studio application.
2. From the Windows section of the New Project dialog box, select Class Library.
3. Name the project MyControls, and click OK.
4. Rename `class1` to `MyWebParts`.

5. Add a reference to the System.Web assembly, and derive the class from the WebPart class as shown in the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI.WebControls.WebParts;
namespace MyControls
{
    public class MyWebPart: WebPart
    {
    }
}
```

6. Add the customizable property for the Web Part that determines the text that is rendered inside the Web Part:

```
[WebBrowsable(true), WebDescription("Displays a custom message"),
WebDisplayName("Display Message"),
Personalizable(PersonalizationScope.User)]
public string ShowMessage
{ get; set; }
```

Note that attributes to the property have also been added.

7. Render the output by overriding the Render method:

```
protected override void
Render(System.Web.UI.HtmlTextWriter writer)
{
    writer.Write(ShowMessage);
}
```

8. Compile the application and place the assembly in the bin directory of the GAC.

9. To use the Web Part, open the ToolBox of your web application, and select Choose Items. Browse to the assembly that you created in the previous step, and click OK. The Web Part will be added to the ToolBox.

10. Drag and drop the MyWebParts control to your web page.

You can now reuse the Web Parts control in your application.

DEVELOPING THE WEB PARTS PAGE

Now that you have created the Web Parts, the next step is to create the Web Parts page. Add two zones, left sidebar and the main zone. The sidebar will display favorite links and the ProcessData User control. The main zone will display the message in the Label control.

To develop a Web Parts page, add a new web page to your web application, and name it WebPartsTest.aspx. In the Design view, drag and drop the WebPartsManager from the ToolBox to the page.

Add a new line and insert a table with one row and two columns in the web page. Add a WebPartZone in the left column of the table. For this Web Part zone, set the ID to MySidebarZone and HeaderText to "This is a sidebar."

Next, add another WebPartZone to the second column with the ID="MainZone" and HeaderText="This is Main Zone." These steps create the structure of the page.

Apart from the Web Part, you also need to add some standard controls to your web page. Drag a Label control from the ToolBox, and drop it in the content area of the MainZone. Notice that this will automatically add the ZoneTemplate element with the Label control

inside it. Add a new attribute to this Label control as its title, and set its value to “Contents” as shown in the following code:

```
<asp:WebPartZone ID="MyMainZone" runat="server" HeaderText="This is the main zone">
    <ZoneTemplate>
        <asp:Label ID="Label1" runat="server" Text="Label" title="Contents">
            <h1>This is a label in the Main Zone.</h1>
        </asp:Label>
    </ZoneTemplate>
</asp:WebPartZone>
```

To the same Web Parts page, add the user control ProcessDataControl that you created earlier, along with a Label as shown in the following code:

```
<asp:WebPartZone ID="MySidebarZone" runat="server" HeaderText="This is a sidebar">
    <ZoneTemplate>
        <asp:Label runat="server" id="linksPart" title="My Links">
            <a href="http://www.asp.net">ASP.NET site</a>
        </asp:Label>
        <uc1:ProcessDataControl ID="control1" runat="server" />
    </ZoneTemplate>
</asp:WebPartZone>
```

Figure 3-3 shows how your page will look after you implement these design steps.

TAKE NOTE*

You can add Web Parts to your page during runtime. To do this, you need to configure the page with the Web Parts catalog using the CatalogZone class.

Figure 3-3

Web Parts test page



The CatalogZone serves as a primary control in the Web Parts control set for hosting the CatalogPart on a web page. The CatalogZone provides the list or catalog of controls that can be added to the web page. A CatalogPart can only be added to a CatalogZone.

The CatalogZone is visible only when the user switches a web page to catalog the display mode. A catalog can have several types of CatalogPart controls, such as PageCatalogPart, DeclarativeCatalogPart, and ImportCatalogPart. These controls provide references to controls that you have added or closed on the page. For example, the PageCatalogPart control provides references for all the controls that you have terminated and can add once again to your web page.

MORE INFORMATION

Refer to the CatalogPart Class section on MSDN for more information on the CatalogPart controls and their use.

Understanding Web Parts Application Development

The primary UI in a Web Parts application consists of ASP.NET server controls, which are contained within different zones or regions that have a common UI and are created as composite controls derived from the `WebPartZoneBase` class. The features of these controls are defined in the base `WebPart` class.

In a Web Parts application, if you want to add user controls or custom server controls, all you need to do is add them to the `WebPartZoneBase` zone. When your page gets compiled, any server control that is not derived from the `WebPart` class is wrapped into an instance of the `GenericWebPart` class. Thus, the server control becomes the child control of the `GenericWebPart` instance. Because the `GenericWebPart` class is derived from the `WebPart` class, it provides the full functionality of the Web Parts.

TAKE NOTE *

User controls and custom controls cannot be derived from Web Parts, and .NET doesn't support multiple inheritances. So, if you want to use user controls and custom controls as Web Parts, you have to use the `GenericWebPart` class.

You can use web parts controls outside your Web Parts application where they function like normal server controls and lose the Web Parts features.

When server controls (that are not derived from the `WebPart` class) are added to the `WebPartZoneBase` zone, you need not use any special declarations, tools, or techniques. You can use the catalog features that enable you to add server controls to a catalog from which users can select and add controls to their web page at runtime. You can make use of the `DeclarativeCatalogPart` and `ImportCatalogPart` classes for this purpose. To add server controls to a zone, you can use the `AddWebPart` method of the `WebPartManager` class.

You can use any server control in your Web Parts application. However, deciding whether to use a server control or a Web Parts control is an important aspect. Consider using server controls and make them a part of the Web Parts control when your application does not require overriding the control's properties and behavior.

When creating your own Web Parts control, consider overriding the following properties:

- **AllowClose:** A value that refers to a user's ability to close a Web Parts control on a page.
- **AllowConnect:** A value that refers to a user's ability to form connections with other controls.
- **AllowHide:** A value that refers to a user's ability to hide a Web Parts control.
- **AllowMinimize:** A value that refers to a user's ability to minimize a Web Parts control.
- **AllowEdit:** A value that refers to a user's ability to edit a Web Parts control.
- **AllowZoneChange:** A value that refers to a user's ability to move the Web Parts control to different zones.

MORE INFORMATION

There are other properties of web part controls in addition to the ones listed. Refer to MSDN for more information about these properties.

TAKE NOTE *

You must add one `WebPartManager` control to every web page that uses web parts controls. This control works with authenticated users only.

Using WebPartManager and WebZones

The `WebPartManager` class manages the Web Parts in the Web Parts application. It also manages the functionality and the events that occur on a web page. This class is responsible for various tasks, such as adding or removing web part controls, managing Web Parts, administering connections between controls, switching between page views, raising events, and importing and exporting controls.

The `WebPartManager` class is derived from the `Control` class and implements the `IPersonalization` and `INamingContainer` interfaces.

WebZones, or **zones**, are UI structural components that function like layout managers on the Web Parts page. Zones organize and contain the controls that are derived from the `WebPart`

class. They also provide the modular page layouts facility in horizontal or vertical orientation. In addition, zones provide common UI elements, such as header/footer styles, titles, border styles, and action buttons for each contained control. These common elements are called the chrome of the control.

USING BUILT-IN WEB PARTS

ASP.NET provides two built-in Web Parts, GenericWebPart and ProxyWebPart. These are derived from the **WebPart** class.

TAKE NOTE*

The **GenericWebPart** class does not declare any markup for the page and can be accessed programmatically.

CERTIFICATION READY?

Customize the layout and appearance of a web page.

7.1

TAKE NOTE*

When the **ProxyWebPart** class replaces the original Web Part that serves either as a consumer or provider for the connection, the connection is not activated or deleted. When the original Web Part is available, the connection gets reactivated.



CREATE A USER CONTROL

Create a user control that asks users for their username and password. Clicking the Login button should raise an event to the web page in which the control is placed:

1. Create the user control "UserLoginControl."
2. Add text boxes for username, password, and a Login button using the following code:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="UserLoginControl.aspx.cs"
Inherits="UserControlTest.UserLoginControl" %>
<asp:Label ID="lblUserName" runat="server" Text="User Name:"/>
    &nbsp;<asp:TextBox ID="txtUserName"
runat="server" Text=""></asp:TextBox>
<br />
<asp:Label ID="lblPassword" runat="server"
Text="Password:"></asp:Label>
    &nbsp;&nbsp;&nbsp;
<asp:TextBox ID="txtPassword" runat="server"
TextMode="Password"></asp:TextBox>
    &nbsp;<br /><br />
```

```
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;
<asp:Button ID="btnLogon" runat="server"
Text="Login" onClick="btnLogon_Click" />
```

- 3.** Register the delegate in the code-behind file and raise the event. Add the following code at the top before the namespace in the code-behind file for the user control:

```
public delegate void LoginUserHandler(string userid, string
password);
// Raise the event on the button click.
if (LoginUser!=null)
{
    LoginUser(txtUserName.Text.Trim(),
    txtPassword.Text.Trim());
}
```

- 4.** Add the Login.aspx page, and add the following code in the .aspx file source at the top of file:

```
<%@ Register Src="~/UserLoginControl.ascx" TagName="UserControl1"
TagPrefix="uc1" %>
```

- 5.** Add the user control inside the <div> tag using the following code:

```
<uc1:UserControl1 ID="LoginControl" runat="server" />
```

- 6.** In the code-behind file of Login.aspx, add the handler for the LoginUser in the Page_Load method:

```
this.LoginControl.LoginUser += new
LoginUserHandler(LoginControl_LoginUser);
if (IsPostBack)
    this.LoginControl.Visible = false;
```

Here is the code for the LoginUser method:

```
void LoginControl_LoginUser(string userid, string password)
{
    if (userid.Length > 0 && password.Length > 0)
    {
        Response.Write("You've logged in successfully.");
    }
    else
    {
        Response.Write("Login failed.");
    }
}
```

The user control is created.

SKILL SUMMARY

In this lesson, you learned about user controls that are useful when you want to provide a consistent look, feel, and layout for your web page controls. You learned that user controls can be used in multiple web pages. You also learned how to create these user controls and use them in web pages. In addition, you learned how to access data from the user controls and position the controls within your web pages with the help of a Panel control. This lesson also explained the events of user controls, how to load them dynamically, and how to raise events to the web page. You also learned to create and use templated user controls.

SKILL SUMMARY

Next you learned about custom controls and how to create your own custom controls by deriving them from the `WebControl` class. You also learned to use the `Render` and `RenderContent` methods of custom controls. After this, you learned to add a custom control to the Web Designer toolbox, to individualize custom controls, and to create a custom designer. Next, you learned to create and use composite custom controls and templated custom controls.

The last section of this lesson explained how you create personalized Web sites that allow users to personalize the content and layout and that allow export and import using Web Parts controls. In this section, you also learned about the `WebPartManager` class and the different Web Parts zones. In addition, you learned how to create Web Parts controls and use them in a web page. You also learned about the important aspects of a Web Parts application and the procedure to develop this application. At the end of the last section, you learned about the `WebPartManager`, `WebZones`, the built-in zones, and the Web Parts in ASP.NET.

For the certification examination:

- Create and add a user control to a web page and ToolBox and raise events
- Create and consume custom Web controls, such as template and composite.
- Create and use Web Parts and Web Part zones.

■ Knowledge Assessment

Fill in the blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. The _____ directive is used to register the user control.
2. To retain the style-rendering logic of a control, you should override the _____ method for writing content.
3. To apply a custom designer to your custom control, you need to create a class that inherits the _____ class.
4. A composite control is similar to the user control; however, it does not provide the designer interface and the _____ file.
5. _____ controls are used for separation of business data from logic.

True / False

Circle T if the statement is true or F if the statement is false.

- | | | |
|---|---|--|
| T | F | 1. Templated user controls are only reusable in the Web site in which the .ascx file is present. |
| T | F | 2. To create your own composite control, you need to create a class that is derived from the <code>CompositeControl</code> class and override the <code>CreateChildControls</code> method. |
| T | F | 3. User controls are best suited for dynamic output. |
| T | F | 4. The <code>EditorZone</code> control plays the role of the base class for the specialized editor controls. |
| T | F | 5. <code>WebPartManager</code> is a sealed class. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. The `UserControl` class is derived from which of the following classes?
 - a. `WebControl`
 - b. `CompositeControl`
 - c. `HierarchicalDataBoundControl`
 - d. `TemplatedControl`
2. Which of the following controls is useful for positioning user controls?
 - a. Panel control
 - b. Button control
 - c. TextBox control
 - d. Menu control
3. Which of the following methods is used while loading user controls dynamically?
 - a. `Page_Load`
 - b. `LoadControl`
 - c. `Page_Init`
 - d. `Render`
4. What are WebPartZones?
 - a. They are the Web Parts UI.
 - b. They are UI structural components.
 - c. They are personalization components.
 - d. They are regions of the web application.

Review Questions

1. What is the importance of templated user controls?
2. List two primary differences between user controls and web pages.
3. Why do you use custom controls?
4. What are Web Part controls?
5. Why do you need to use the `RenderControls` method and not the `Render` method while deriving a control from the `WebControl` class?

■ Case Scenarios

Scenario 3-1: Using User and Custom Controls

Selecting between web user controls and custom web controls is an important decision in developing web applications in ASP.NET. Consider a situation where you need to display a navigation bar with Back and Next buttons on your Web site. You also need to display a control that will prevent users from entering alphabetical and special characters. How will you develop this application?

Scenario 3-2: Using Composite Controls

You need to display the list of countries in a List control on a web page. When the user selects a country from the list, the selected country should be displayed in a label. How will you achieve this?



Workplace Ready

The Importance of Controls

You are developing a web application for an inventory control system. The web application consists of various web pages that allow users to enter information, such as the available stock, the stock ordered, the items sold, the price of each item, etc. Your aim is to provide a consistent interface and behavior for all the screens that allow users to enter data. Additionally, your data-entry screen should also ensure data validation. For example, the item sold must be a positive number; the price of an item must be a valid price, such as a positive value, etc. How do you achieve this?

Manipulating Data Using ADO.NET

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Understanding ADO.NET.	Manipulate data by using DataSet and DataReader objects.	3.2
Using ADO.NET Connected Classes.	Manipulate data by using DataSet and DataReader objects.	3.2
Using ADO.NET Disconnected Classes.	Manipulate data by using DataSet and DataReader objects.	3.2
Manipulating Data Using DataSet, DataAdapter, and DataReader Objects.	Manipulate data by using DataSet and DataReader objects.	3.2

KEY TERMS

cursor
database
foreign key

primary key
resultset
table

Most applications use **databases** to store data. You can access data from these databases and display the data in your application's user interface. Therefore, when you design an application, you first design the data model and subsequently create the application on top of the data model. ADO.NET is a data model and is part of Microsoft.NET that provides data access from different data sources, such as SQL Server, Oracle, and Microsoft Access. ADO.NET has evolved from ADO, which has been a very successful object model for writing data-aware applications.

This lesson describes how to use ADO.NET in connected and disconnected modes. In addition, the lesson explains how you can read and manipulate data using the **DataReader** and **DataSet** objects. It also explains how you can create and use database connections using the **ObjectDataSource** and **SqlDataSource** controls.

■ Understanding ADO.NET



ActiveX Data Objects (ADO) is a collection of ActiveX objects that are meant to work only in a connected environment. As a result, your application can access data only while it is connected to the database. ADO allows data access from SQL as well as non-SQL databases. ADO.NET is an improved version of ADO. ADO is analogous to ADO.NET, which enables you to work in a disconnected environment. This means that you can disconnect from the database and fetch data from the database into the containers called **DataSets** provided by ADO.NET. In this case, to exchange data with the database, you can establish the connection with the database as required.

It is important to remember that ADO.NET is not a collection of ActiveX objects and that the letters A, D, and O in ADO.NET do not stand for ActiveX Data Objects. In fact, you will be surprised to know that these letters mean nothing. Microsoft simply decided to continue with the abbreviation ADO so that it is not confusing for programmers shifting from ADO to ADO.NET.

ADO.NET utilizes the data providers of the .NET Framework for connecting to a database, executing commands, and retrieving results. The data manipulation and data access functionalities of ADO.NET are explicitly segregated into discrete components that can be used independently or together, like DataReaders, DataAdapters, and DataSets.

Before discussing connected and disconnected modes in detail, it is important to understand data providers.

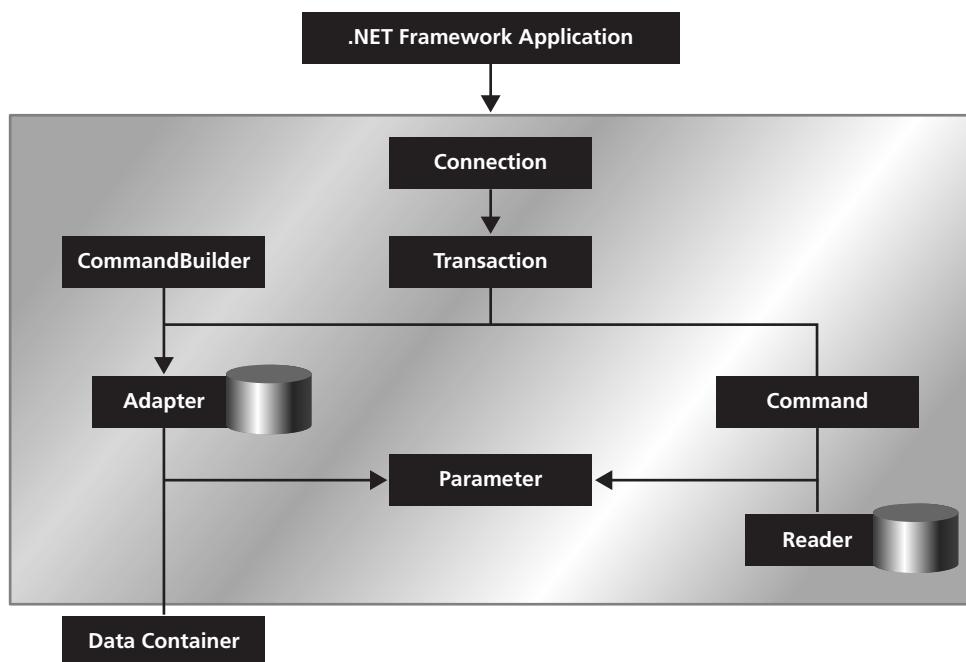
Understanding Data Providers

Data providers enable you to connect to the data source, allow access to the data, and enable you to manipulate it.

Data obtained using a data provider is either processed directly or placed in an ADO.NET DataSet object. This DataSet object can be exposed to the user—in an ad-hoc manner or on-demand basis—without connecting to the data source, as long as the DataSet holds the latest data. This data can be combined with data from multiple sources, or it can be accessed from a remote location.

The .NET managed provider is the key element in the ADO.NET architecture. It is made up of a smaller set of interfaces and has simple data access architecture based on the .NET Framework 3.5. Figure 4-1 shows the ADO.NET architecture.

Figure 4-1
ADO.NET architecture



The .NET managed provider establishes the connection with the database to retrieve and modify data as required by the application. With the help of the datatypes defined in the .NET Framework, the classes in the managed provider interact with the defined data source to return application-specific data to the application. Figure 4-2 shows the interconnection between the components of the managed provider.

Figure 4-2

Interconnection between the objects of the managed provider

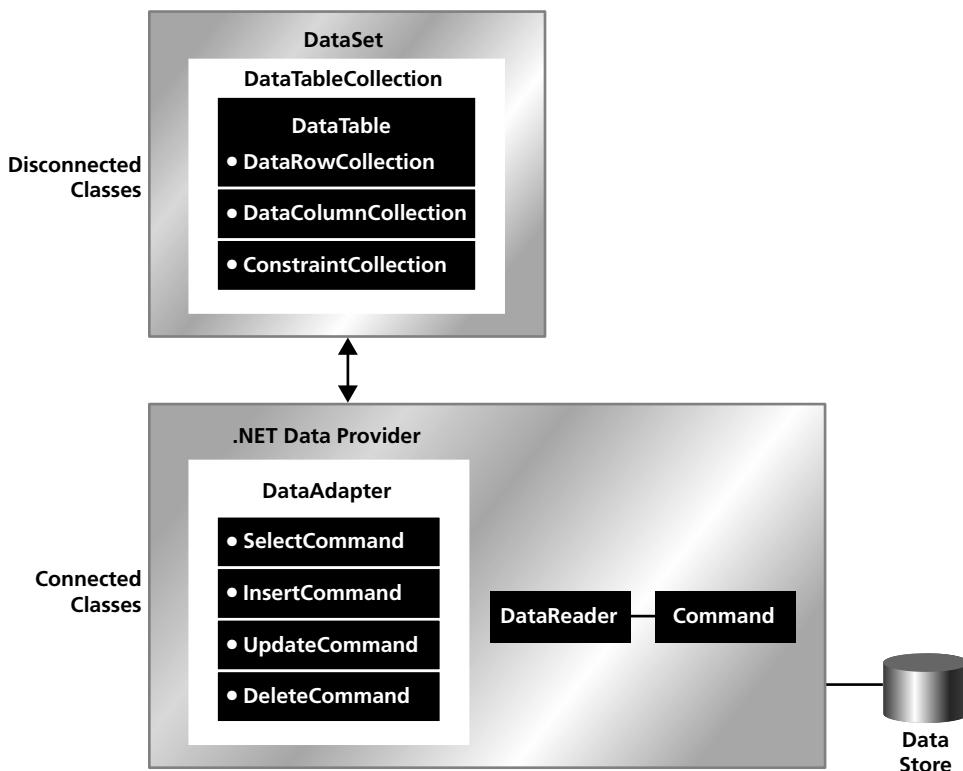


Table 4-1 lists the objects of the managed data providers of the .NET Framework.

Table 4-1

Objects of the .NET managed data providers

OBJECT	DESCRIPTION
Connection	Establishes a connection with the data source like SQL server and MS SQL. The base class of all Connection objects is DbConnection .
Command	Executes commands on data sources to read or manipulate data from the front end. It can be executed within the boundaries of a Connection object's transaction. It can also have parameters. DbCommand is the base class for all the Command objects.
DataReader	Reads data from the data source in a forward-only and read-only fashion. DbDataReader is the base class for all the DataReader objects.
DataAdapter	Populates the data source and updates the data source with the modified DataSet data. DbDataAdapter is the base class for all DataAdapter objects.
Transaction	Uses all the commands required for a data source transaction. You can have an INSERT , UPDATE , or DELETE operation as transactions. DbTransaction is the base class for all the Transaction objects and uses System.Transaction namespace of ADO.NET.
CommandBuilder	Generates command properties of a DataAdapter or can obtain parameters from the stored procedures. The DbCommandBuilder is the base class for all the ComamndBuilder objects.
ConnectionStringBuilder	Builds the connection string to be used in the Connection object. DbConnectionStringBuilder is the base class for all ConnectionStringBuilder objects.

Table 4-1

OBJECT	DESCRIPTION
Parameter	Defines the values for input, output, and return values required by the stored procedure and commands. <code>DbParameter</code> is the base class for all <code>Parameter</code> objects.
Exception	Obtains the error encountered while interacting with the data source. The .NET Framework exception occurs when a client application encounters an error. <code>DbException</code> is the base class for all <code>Exception</code> objects.
Error	Identifies the cause of the error while retrieving the error information.
ClientPermission	Is required to access the security attributes of the .NET Framework data provider. <code>DbDataPermission</code> is the base class for all <code>ClientPermission</code> objects.

To transfer data between the data source and the application, the ADO.NET library uses provider classes. These classes work as a bridge between the application that is accessing the data and the data source. ADO.NET provider classes include `DbProviderFactory` and `DbProviderFactories`. Table 4-2 describes the managed data providers that the .NET Framework provides:

Table 4-2

Managed data providers in the .NET Framework

DATA PROVIDER	NAMESPACE	DESCRIPTION
OLEDB	<code>System.Data.OleDb</code>	Classes in this data provider provide data access to data sources such as SQL Server 6.5 and earlier versions, SyBase, Microsoft Access, and DB2/400.
ODBC	<code>System.Data.Odbc</code>	Classes in this data provider provide data access to data sources when no other provider is available.
SQL Server	<code>Microsoft.SqlServer.Server</code>	Classes in the SQL Server data provider provide functionality similar to the OLEDB provider, except that these classes are tuned for data access from SQL Server 7.0 and later versions.
Oracle	<code>System.Data.OracleClient</code>	The Oracle data provider classes offer data access for Oracle 8i and later servers. They perform better than OLEDB.

DB2 and MySQL have third-party data providers that can be easily downloaded from the web. A discussion about ADO.NET would not be complete without a mention of transactions. A transaction is a collection of DML statements that form a logical unit of work. A database transaction must be atomic, consistent, isolated, and durable (acid). SQL transactions start with a `START TRANSACTION` statement. Transactions end with a `COMMIT` operation when the execution is successful. If an error occurs, then the transaction ends with a `ROLLBACK`, which is the benefit of transaction. A .NET transaction can be written on the front end or in the application block when a SQL transaction is written in the stored procedure.

TAKE NOTE *

The .NET Data Access Application block provided by the Enterprise Library allows you to reuse software components and reduce the amount of custom code that you write to create, test, and maintain stored procedures. This .NET component contains optimized data access code that calls stored procedures and runs SQL text commands.

■ Using ADO.NET Connected Classes



THE BOTTOM LINE

In the connected mode, you access data by setting up a connection and executing commands programmatically or with the help of data source controls. This means that the application can access data only while it is connected to the database.

Working with the Connection Objects

To connect to a data source from within an ADO.NET application, you use **Connection** objects.

Using these objects, you first open the connection, process the transactions on the database, and then release the connection by closing it.

You open a connection by first instantiating the **Connection** object. For example, to connect to Microsoft SQL Server 7.0 or later, you use the **SqlConnection** object of the .NET Framework Data Provider for SQL Server. The **SqlConnection** object is the provider-specific connection object that inherits the properties and methods from the **DbConnection** base class.

Consider a drugstore that needs to maintain records of drug sales, purchases, and stock. It uses the DrugStore database on SQL Server to store data. Suppose that **tables** already exist in the database for DrugVendor, DrugStock, and Invoice.

To get the records of available drug stocks, you first need to open a connection. For this, you need to create a **Connection** object and then assign a connection string to it. A connection string specifies the database name and user credentials with which you want to connect to the database. To connect to the DrugStore database, you need to specify a valid connection string to create a connection as shown here:

```
SqlConnection connection = new SqlConnection();
connection.ConnectionString = "Server=.;Database=DrugStore;Trusted_
Connection=true"; connection.Open();
connection.Close();
```

In this code, the **Connection** object is a **SqlConnection** datatype and an instance of the **SqlConnection** class. These classes belong to the **System.Data** namespace. The **ConnectionString** property of the **Connection** object is set to use the DrugStore database present in the local computer by using a period ("."). To use the trusted SQL Server authentication mode to connect to the data source, you set **Trusted_Connection** to True. You can also use the user ID and password for authentication. You will learn more about authentication modes in the next section.

To use an established connection, you need to first open it using the **Open** method of the **Connection** object. After using the connection, you need to close it using the **Close** method.

Alternatively, you can use the **Dispose** method to dispose of a connection. To force the execution of the **Dispose** method, you may place it under the **using** block:

```
using (connection)
{
    connection.Open();
    // Remaining code for data access goes here
}
```

TAKE NOTE *

The **Close** method closes the connection temporarily; you can open the connection again using the same instance. However, the **Dispose** method permanently destroys the **Connection** object. So, it is a good practice to use the **Dispose** method in the **Finally** block of the code so that all the objects created within the class are released. However, you should note that coding styles may differ; and based on the way you write the code, you may not dispose of the connection unless you dispose of the object that has created the connection.

The connection strings used to define `Connection` objects are independent of programming languages. However, these strings are configured in different ways for each .NET provider.

CONFIGURING CONNECTION STRINGS IN ODBC

Open Database Connectivity (ODBC) is required to connect to a database that has ODBC drivers. This is the old method of connecting to a database, but it is still supported by the .NET Framework. To establish a connection to a database using ODBC, the connection string for a connection object should be defined with the parameters listed in Table 4-3.

Table 4-3

Parameters for the ODBC connection string

PARAMETER	DESCRIPTION
DSN	Is the data source name that can be configured from Control Panel > Administrative Tool > Data Source (ODBC).
Server	Is the name of the server with which the connection is to be established.
Trusted_Connection	Describes the connection string to specify the security type. When using the domain account, this connects the logged-on user. The user ID and password parameters are optional. You need these parameters when you want to pass credentials of a user other than the one who is logged on.
Database	Is the name of the database from which data is to be retrieved or in which data should be updated.
DBQ	Specifies the physical path of the data source.

To open a database connection by specifying a connection string for ODBC connectivity with an appropriate username and password, use the following code:

```
using System.Data;
using System.Data.Odbc;

protected void btnODBC_Click(object sender, EventArgs e)
{
    Label lb1 = new Label();
    Label lb2 = new Label();
    OdbcConnection conn = new OdbcConnection();
    conn.ConnectionString = @"dsn=DrugStore";
    conn.Open();
    lb1.text = "ODBC connection established successfully";
    this.Controls.Add(lb1);

    conn.Close();
    lb2.text = "ODBC connection closed successfully";
    this.Controls.Add(lb2);
}
```

The example shows you how an ODBC connection can be established. Here a button is added to the page, and the ODBC connection is established on the click of the button. The result is displayed in the labels. First, the code creates a `Connection` object of type `ODBCConnection` and then it defines the connection string for the `Connection` object. Finally, it opens and closes the connection.

CONFIGURING AN OLEDB CONNECTION STRING

Apart from ODBC, a common method used to access databases is Object Linking and Embedding for Databases (OLEDB). Similar to other connecting methodologies, this method also requires a connection string with specific parameters. Table 4-4 lists the parameters of an OLEDB connection string.

Table 4-4

Parameters for the OLEDB connection string

PARAMETER	DESCRIPTION
Data Source	Indicates the physical location of the database, or the database name.
File Name	Denotes the physical location of the file containing the connection information.
Persist Security Info	Specifies whether the security information from the actual connection is to be retained. If the value of this keyword is set to <code>true</code> , the security information in the connection string that was originally provided can be reused. If set to <code>false</code> , the security information will be dropped from the original connection string.
Provider	Denotes the vendor-specific driver used for connecting to a data store.

The next code shows you how to connect to a database using OLEDB:

```
using System.Data;
using System.Data.OleDb;protected void btnOLEDB_Click(object sender,
EventArgs e)
{
    Label lb1 = new Label();
    Label lb2 = new Label();
    OleDbConnection conn = new OleDbConnection();
    conn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=c:\Program data\DrugStore.mdb";
providerName="System.Data.OleDb";

    conn.Open();
    lb1.text = "OLEDB connection established successfully";
    conn.Close();
    lb2.text = "OLEDB connection closed successfully";
    this.Controls.Add(lb1);
    this.Controls.Add(lb2);
}
```

The example shows you how an OLEDB connection can be established. Here a button is added to the page and an OLEDB connection is established on the click of the button. The result is displayed in the labels. First, a `Connection` object of type `OleDbConnection` is created, and then, the connection string is defined for the `Connection` object. Finally the connection is opened and closed.

In this code, the connection string uses the settings stored in a universal data link (.udl) file, `MyAppData.udl`. Microsoft Data Access Components provide the .udl files to store connection information. These files are provided in Windows 2000 and later versions. The OLEDB connection string uses the Jet driver, which is the Access driver, and opens the DrugStore database file. Retrieving the connection string from the connection returns the connection that was originally passed in, without the security information.

CONFIGURING SQL SERVER CONNECTION STRINGS

One commonly used database is the SQL Server database. Often, you will need to establish connections to SQL database servers. This can be achieved with the help of SQL Server data providers. These providers connect to SQL Server 7.0 and later versions. Table 4-5 describes some common SQL Server connection string parameters.

Table 4-5

Parameters for SQL Server connection strings

PARAMETER	DESCRIPTION
Data Source, Addr	Specifies the name or IP address of the database server to which a connection should be established.
Failover Partner	Indicates the provider that is used to mirror databases in SQL Server 2005.
AttachDbFilename, extended properties, Initial File Name	Indicates the full or relative path and the name of the database to which a connection should be established. The path can contain the keyword <code>IDataDirectory</code> , indicating the data directory of the application to which the connection will point.
Integrated Security, Trusted_Connection	Carries the values <code>true</code> , <code>false</code> , or <code>sspi</code> . The default value is <code>false</code> . When set to <code>true</code> , it means the connection to the SQL Server is secure, and that the user's domain account is responsible for authentication to this server.
Persist Security Info, persistsecurityinfo	If set to <code>true</code> , this parameter retrieves the connection string that was provided earlier. However, if set to <code>false</code> , the connection string contains all the information originally provided, without the security information. The default value is <code>false</code> .
User ID, UID, User	Specifies the username that you need to use to connect to the SQL Server. This parameter should be specified when a trusted connection is not established.
Password, Pwd	Specifies the password that you need to use to connect to the SQL Server. This parameter should be specified when a trusted connection is not established.
Pooling	If this Boolean parameter is set to <code>true</code> , all new connections are established from the pool. However, if the pool does not exist, a new pool is created.
Max Pool Size	Indicates the maximum number of connections that can be appended to a connection pool. The default value is <code>100</code> .
Min Pool Size	Indicates the minimum number of connections that should be present in a connection pool. The default value is <code>0</code> .
Asynchronous Processing, Async	If this parameter is set to <code>true</code> , execution of asynchronous commands on a connection is enabled. The default value is <code>false</code> .
Connection Reset	If this parameter is set to <code>true</code> , a database connection is automatically reset when it has been removed from the pool. The default value is <code>true</code> .
Connection Timeout	Specifies the maximum time (in seconds) that the application can wait to connect to a data store. The default value is <code>15</code> seconds.
Connection Lifetime	Specifies the maximum time (in seconds) that a connection can be active.
Load Balance Timeout	Specifies the minimum duration (in seconds) that a connection should be active. The default value is <code>0</code> .
Packet Size	Specifies the size (in bytes) of each data packet sent to the SQL Server. The default value is <code>8,192</code> .
Workstation ID, Wsid	Indicates the name of the client computer that is connecting to the SQL Server.
Server	Denotes the name of the server that the SQL server you want to connect to is running. This parameter can accept either the name of the server or the name of the SQL server client network utility. If you want to connect to the SQL server running on your local machine, you need to assign the value <code>Local</code> .

MORE INFORMATION

Security Support Provider Interface (SSPI) is an application provider interface that provides security to applications. It is the basic user authentication interface for many back office applications, which require single sign on and generation and verification of digital signatures. To know more about SSPI, refer to the section on SSPI on MSDN. For more information on the other parameters of the SQL Server connection string, refer to MSDN.

Here is an example of using `SqlConnection` to connect to the SQL Server database with the help of SQL Provider:

```
using System.Data;
using System.Data.SqlClient;

protected void btnSQLConn_Click(object sender, EventArgs e)
{
    Label lb1 = new Label();
    Label lb2 = new Label();
    string SQLString;
    SQLString = "Data Source=Local;Initial Catalog=DrugStore;Persist Security Info=True;User ID=sa;Password=sapass;";
    SqlConnection Conn = new SqlConnection(SQLString);
    try
    {
        Conn.Open();
        lb1.Text = "SQL connection established successfully";
        // The following syntax will add a label at runtime on the page. Note that you can also create the label at design time.
        this.Controls.Add(lb1);
        // Write code here to access or manipulate data using this connection string.
        Conn.Close();
        lb2.Text = "SQL connection closed successfully after use";
        this.Controls.Add(lb2);
    }
    catch (SqlException exp)
    {
        // Handle the error
        lb1.Text = "Error" + exp;
    }
}
```

TAKE NOTE*

As already mentioned, if you need to connect to databases in SQL Server 6.5 or earlier versions, you have to use the OLEDB provider.

In this code, similar to the two ASP.NET data provider examples, the connection with SQL Server database is established using the `SqlConnection` object of SQL Provider. It is always advisable to open the connection within the `try catch` block. This is useful in case any error occurs while connecting to the database. The error can be trapped and rectified.

CONFIGURING CONNECTION STRINGS IN WEB CONFIGURATION FILES

Storing connection strings in the `web.config` file helps make simple changes to the string without touching the code. Therefore, the code need not be recompiled. The configuration can also be stored in the machine configuration file. To store the connection string in the `web.config` file, you need to use the `<connectionString>` tag as shown in the following code:

```
<connectionStrings>
<clear />
<add name="dbDrugStore"
providerName="System.Data.SqlClient" connectionString=
"Data Source=.\SQLEXPRESS;
AttachDbFilename=|DataDirectory|DrugStore.mdb; Integrated
```

```
Security=True; User Instance=True"/>
</connectionStrings>
```

The `<clear>` element clears all the connection string references that may be inherited from the collection stored in the `<connectionString>` element of machine.config. Therefore, the connection string added within the `<add>` element will be considered.

In the previous code, the connection string is first declared in the web.config file, and connection can be created whenever required in the code.

The web.config file stores the `ConfigurationManager` collection, which in turn stores the static `ConnectionStrings` collection. The next code shows you how to read multiple connection strings from the web.config file and display them in a label:

```
using System.Configuration;
public partial class ConnectionPage: System.Web.UI.Page
{
    string DBConn;
    protected void Page_Load(object sender, EventArgs e)
    {
        Label lb1 = new Label();
        DBConn = ConfigurationManager.ConnectionStrings
        ["WebConnectionString"].ConnectionString;
        lb1.Text = DBConn;
        this.Controls.Add(lb1);
    }
}
```

For better understanding, you can also display connection strings in different labels.

USING CONNECTION POOLING

Connection pooling is the process of queuing existing connections and reusing them whenever a request to establish new connections is raised. This process increases the efficiency of the application because new connections need not be created every time a data request is placed. All connections that you create get queued up in the connection pool on encountering the Close command. So, when another application wants to establish a connection next, the connection will not be created again and will be directly retrieved from the pool.

ASP.NET provides a connection manager to implement connection pooling. This connection manager is responsible for checking all the available pools for a connection when a new connection is requested. If found, the same connection is used for data retrieval. If not found, a new connection is created and added to the pool. Adding connections to the pool depends on the size of the pool. In situations when the pool is already full, the new connection requests are queued and processed when the pool has the necessary space.

ADO.NET manages several pools at a time for each configuration. Connections are separated into pools by the connection string; in the case of integrated security, the windows identity is used. Connections are also pooled based on whether they are enlisted in a transaction.

In the code example given next, three connections are being created with two pools to serve the requests. The first and the second connection strings contain the same configuration and, therefore, they are added to one pool. The third connection is added to the second pool:

```
using (SqlConnection connection = new SqlConnection ("Integrated
Security=SSPI;Initial Catalog=Northwind"))
{
    connection.Open();
    // Pool A is created.
}
```

```

using (SqlConnection connection = new SqlConnection ("Integrated
Security=SSPI;Initial Catalog=pubs"))
{
    connection.Open();
    // Pool B is created because the connection strings differ.
}
using (SqlConnection connection = new SqlConnection ("Integrated
Security=SSPI;Initial Catalog=Northwind"))
{
    connection.Open();
    // The connection string matches pool A.
}

```

There are certain guidelines that you need to follow when implementing connection pooling in your application. The primary requirement is that the string of all connections that are accumulated in a pool should be exactly the same. These strings are case sensitive. In addition, the process IDs of all connections should also be the same. A connection pool cannot queue connections for multiple processes. Additionally, the ID of the user or service that is participating in the pooling process should also be the same.

Connection pooling is a highly streamlined process, which starts with adding connections to a pool and goes on until requests are serviced.

Connection pooling has the following steps:

- 1. Adding the connection:** For a unique connection string, a connection is added in the pool. Connections are added to the pool based on the maximum pool size, and connections are created for the unique connection string based on the minimum pool size. `Connection` objects are created for unique connection strings. For each new `Connection` object request, an available `Connection` object from the pool is identified and returned if it exists. If not, then a new connection is created.

TAKE NOTE *

If the maximum pool size has been reached, and no usable connection is available, the request is queued. The pooler then tries to reclaim any connections until the time-out is reached (the default is 15 seconds). If the pooler cannot satisfy the request before the connection times out, an exception is thrown.

- 2. Removing a connection:** When a connection is found idle for a stipulated period of time, or if its connection with the server has been severed, the connection pooler removes the connection from the pool. Note that a severed connection can be detected only after attempting to communicate with the server. If a connection is no longer found connected to the server, it is marked as invalid. Invalid connections are removed from the connection pool only when they are closed or reclaimed. If a connection exists to a server that has disappeared, this connection can be drawn from the pool even if the connection pooler has not detected the severed connection and marked it as invalid. This is the case because the overhead of checking that the connection is still valid would eliminate the benefits of having a pooler by causing another round trip to the server to occur. When this occurs, the first attempt to use the connection will detect that the connection has been severed, and an exception will be thrown.
- 3. Clearing the connection pool:** There are two methods of the `SqlConnection` class, which is inherited from `DbConnection`, to clear the connections in a pool—`ClearAllPools` and `ClearPool`. With `ClearAllPools`, the connection pool for a given provider is cleared. With `ClearPool`, the connection pool associated with a specific connection is cleared. When clearing a pool, if an active connection is encountered, the link between the connection and the pool is voided, and the pool is cleared. After the connection is closed, it is discarded rather than adding it back to the pool.

After the connection is established, with or without pooling based on the requirement, you need to execute various commands to fetch the data or to modify it. Each data provider has its own set of commands, which are derived from the `IDbCommand` interface.

Working with Command Objects

The ADO.NET object model provides two command objects derived from the `DbCommand` class, namely, `Command` and `DataAdapter` to retrieve data from the data sources. When using SQL Server as database, you can use `SqlDataAdapter` to improve the performance along with `SqlCommand` and `SqlConnection`.

You can use the `SqlCommand` object to execute stored procedures and return a *cursor*. `SqlCommand` is used by SQL Server database and belongs to SQL Data Provider.

X REF

Note that `DataAdapter` will be covered in detail in the next section of the lesson.

The `DataAdapter` object retrieves data from a data store and loads it into the data containers; that is, the `DataSets` and `DataTables`. There is one major difference between `SqlCommand` and `DataAdapter`—`SqlCommand` is a traditional way of command execution and usually returns a cursor that traverses through the rows in a table. The connection will be busy as long as the cursor is in use. On the other hand, `DataAdapter` is more efficient because after loading the data in a `DataSet`, the application can process the data without connecting to the data source.

USING THE SQLCOMMAND CLASS

The `SqlCommand` class is derived from the ADO.NET `DbCommand` and implements the `IDbCommand` interface. This class represents a SQL Server statement or a stored procedure. The command is always executed on an existing connection and optionally on a transaction that is represented by the constructor in the class. The definitions of the overloaded constructors of the `SqlCommand` class are:

```
public SqlCommand();
public SqlCommand(string);
public SqlCommand(string, SqlConnection);
public SqlCommand(string, SqlConnection, SqlTransaction);
```

As shown in the first definition, a `SqlCommand` object can be executed without passing any parameter. The string parameter passed in the second definition denotes the name of the stored procedure to be executed. By using `SqlConnection` as shown in the third definition, the connection object used can be passed as a parameter. Similarly, by using `SqlTransaction` as shown in the fourth definition, the transaction context on which the command runs can also be passed as parameter. Note that you need to explicitly establish the connection for the command and assign it to the command because the command object does not open a connection on its own.

The code that follows reads the order data from the `ProductOrder` table using the `SqlCommand` object and displays the result on the screen using the `response.write` command. You can see how the `SqlConnection` is created and the `SqlCommand` object is created and executed after opening the `SqlConnection`. The `SqlDataReader` object is used to read the data:

```
private void ReadOrderData(string connectionString)
{
    string queryString;
    queryString = "select * from ProductOrder;";
    using (SqlConnection connection = new
SqlConnection (connectionString))
    {
        SqlCommand command = new SqlCommand(queryString, connection);
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Response.Write(String.Format("{0}, {1}", reader[0],
reader[1]));
        }
        reader.Close();
    }
}
```

To execute a stored procedure, the `CommandText` property of the `Command` object is assigned with the name of the stored procedure, and the `CommandType` property is set to show that this is a call to a stored procedure. You can set the following values for the `CommandType` property:

- **Text:** Indicates that `CommandText` is of type Transact-SQL and that the text is to be executed directly. Transact-SQL is the default type for this property.
- **StoredProcedure:** Indicates that the content of the `CommandText` property is the name of the stored procedure that must be executed in the target database.
- **TableDirect:** Indicates that the property of the `CommandText` is a comma-separated string of table names. This command is only used with the OLEDB data provider. It returns all the columns and rows of the tables listed.

TAKE NOTE*

You can find the usage of the other `Execute` methods, such as `ExecuteNonQuery`, `ExecuteQuery`, `ExecuteScaler`, and `ExecuteReader` at relevant locations throughout the lesson.

The following code example shows you how to execute a command using the `StoredProcedure` command type:

```
using (SqlConnection conn = new SqlConnection(ConnString))
{
    SqlCommand cmd = new SqlCommand(sprocName, conn);
    cmd.CommandType = CommandType.StoredProcedure;
    cmd.Connection.Open();
    cmd.ExecuteNonQuery();
}
```

USING THE DBPARAMETER OBJECT

The `DbParameter` object is used to pass data to the data store when executed with the `Command` object with commands. For example, you can specify the `DbParameter` string with `SQLCommand`.

An example of a parameterized command is:

```
SqlParameter parm = new SqlParameter();
parm.ParameterName = "@Product_ID";
parm.DbType = DbType.Int;
// ParameterDirection.Input helps in passing the input parameter of
// specified length.
parm.Direction = ParameterDirection.Input;
cmd.Parameters.Add(parm);
```

In this code, `SqlParameter` is the parameter class for the SQL server managed provider.

You can also pass parameters in an SQL statement, as shown in this example:

```
SELECT * FROM ProductOrder WHERE Product_ID=@Product_ID
```

The following code configures the `DbCommand` and `DbConnection` objects and passes `@Product_ID` as the parameter:

```
private void ReadOrderData_Parameter(string
connectionString, int Product_ID)
{
    using (SqlConnection connection = new SqlConnection
(connectionString))
    {
        connection.Open();
        //
        string SqlCmd = "spReadOrderbyProduct_Name";
        SqlCommand command = new SqlCommand(SqlCmd, connection);
        command.CommandType = CommandType.StoredProcedure;
        command.Parameters.Add("@Product_ID", SqlDbType.Int);
        command.Parameters["@Product_ID"].Value = Product_ID;

        SqlDataReader reader = command.ExecuteReader();
```

```
        while (reader.Read())
    {
        Response.Write(String.Format("{0}, {1}", reader[0],
reader[1]));
    }
    reader.Close();
    connection.Close();
}
}
```

In this code, first the `SqlConnection` is created as usual, then the `SqlCommand` object is created and assigned the parameter as in the next code:

```
command.Parameters.Add("@Product_ID", SqlDbType.Int);
```

The value of the parameter is set as:

```
command.Parameters["@Product_ID"].Value = Product_ID;
```

The parameter in this case is passed as an argument to the `ReadOrderData_Parameter` function by the caller.

Working with DataReader and DataAdapter Objects

`DataReader` and `DataAdapter` are two more essential objects of the connected architecture of ADO.NET.

The `DataReader` object works in the connected mode and reads data only in the forward direction. You can instantiate the `DataReader` object using the `ExecuteReader` method. The `DataReader` object stores the resultset at the client end in a buffer for the reader to read through. A **resultset** is a set of rows that are generated on execution of a SQL query or statement.

X REF

You will learn more about `DataReader` and `DataAdapter` objects in later sections.

Reading data using `DataReader` improves performance because only one data record is cached at a time, and the wait time is negligible for loading the data in memory.

`DataAdapter` helps perform the insert, update, delete, and select operations on the database with the help of the `SelectCommand`, `UpdateCommand`, `InsertCommand`, and `DeleteCommand` properties. The `DataAdapter` object is specific to the data provider. This is similar to the way each data provider has its own `Connection`, `Command`, and `DataReader` object.

■ Using ADO.NET Disconnected Classes

THE BOTTOM LINE

Using the disconnected classes of ADO.NET, you can access data in the disconnected mode. To do this, you need to use the `DataTable`, `DataSet`, and `DataTableReader` objects. The web application loads the data from the data source into its `DataSet` and `DataTable` objects and processes the data at application level in the disconnected mode using the `DataTableReader` object. By accessing the data in disconnected mode, the performance of the application can be enhanced by reducing the load on the data servers.

Using the DataTable Object

The `DataTable` object represents tabular data stores, which have rows and columns.

Therefore, this object consists of `DataRow`s and `DataTableColumn`s. The unique feature of `DataTable` is that it facilitates creating tables explicitly in the memory while performing disconnected data operations. The `DataTable` class is instantiated while creating the

`DataTable` explicitly. You can add the `DataColumn` object to hold defined data of a datatype and add the rows using the `DataRow` object containing the data. To maintain data integrity, you set constraints on the `DataColumn` object.

The following code shows you how to create a `DataTable` object and add a `DataColumn` object to the data table:

```
private void GenerateDataTable()
{
    DataTable dataTable = new DataTable("Product");
    dataTable.Columns.Add(new DataColumn("Product_ID", typeof(int)));
    dataTable.Columns.Add(new DataColumn
    ("Product_Name", typeof(string)));
    dataTable.Columns.Add(new DataColumn("Product_Price", typeof
    (decimal)));
    DataRow dataRow = dataTable.NewRow();
    dataRow["Product ID"] = 1;
    dataRow["Product Name"] = "Medicine1";
    dataRow["Product Price"] = 100;
    dataTable.Rows.Add(dataRow);

    DataRow dataRow = dataTable.NewRow();
    dataRow["Product ID"] = 2;
    dataRow["Product Name"] = "Medicine2";
    dataRow["Product Price"] = 120;
    dataTable.Rows.Add(dataRow);
    GridView_DataTable.DataSource = dataTable;
    GridView_DataTable.DataBind();
}
```

This code creates the Product table and adds columns to store data for “Product_ID,” “Product_Name,” and “Product_Price.”

You need to understand the following properties of the column, out of which the `DataType` property is used in this code. You can set other properties of the column in a similar way:

- **DataType:** Indicates the datatype of the column. This can be string, short, integer, float, or double.
- **MaxLength:** Specifies the maximum length of the column. A value of -1 indicates that maximum length check need not be performed.
- **Unique:** Specifies if duplicate values are allowed in the column.
- **AllowDBNull:** Specifies if the column allows NULL values. A `false` value indicates that the column needs to have a valid value.
- **Caption:** Specifies the name of the column.

Once you create a table, you can add columns and rows to it. Then, you can set relationships with other tables and assign a **primary key**. Further, you can populate the table with data using the `Add` method or `DataRow` objects. You can also manipulate the data you entered. In the previous example, you have directly created columns by adding their names and defining their datatypes.

You can also explicitly define the property as shown here:

```
// Add the DataColumn using all properties
DataColumn pid = new DataColumn("Product_ID");
pid.DataType = typeof(string);
pid.MaxLength = 10;
pid.Unique = true;
pid.AllowDBNull = false;
pid.Caption = "Product_ID";
product.Columns.Add(pid);
```

TAKE NOTE *

ASSIGNING THE PRIMARY KEY TO COLUMNS

To establish a unique identity for each row, data tables use primary keys. A primary key can be formed from one or more columns. For example, in the Vendor table of a database, a column named VendorID will act as a primary key. Sometimes a primary key can be combination of two columns or more, so that both together form a unique identity for each row. Such a primary key is also called a composite primary key.

The following code shows you how to set the primary key for a column in a single line:

```
DataTable dataTable = new DataTable("Product");

dataTable.Columns.Add(new DataColumn("Product_ID", typeof(int)));
dataTable.PrimaryKey = new DataColumn[] { dataTable.
Columns["Product_ID"] };

dataTable.Columns.Add(new DataColumn("Product_
Name", typeof(string)));
dataTable.Columns.Add(new DataColumn("Product_Price",
typeof(decimal)));
```

When working with disconnected data, it is beneficial to use GUID as the primary key. By using GUID as the primary key, you can:

TAKE NOTE *

- Avoid data migration problems resulting from duplicate key in auto-numbered columns. Duplicate keys may occur when too many users create rows on the data source at the same time before the first transaction is completed.
- Connect multiple tables through relationships and maintain the uniqueness of table data. Using GUID, you can also simplify the maintenance of data because any change to the primary key need not be updated to the related tables.
- Work effectively with joins on tables.

POPULATING DATA USING DATAROW OBJECTS

After creating a table schema, the next step is to add data rows to it. You can do this by using a `DataRow` object. Several `DataRow` objects denoting a collection of data rows comprise to form a `DataTable` object. When creating a `DataRow` object, you need to ensure that it conforms to the column constraints specified by its related `DataTable` object.

To insert data rows into a table, you can use the `Add` and `Load` methods. The `Add` method adds `DataRow` objects to the `Rows` collection of the `DataTable` object. An overload of the `Add` method accepts an array of objects instead of a `DataRow` object. These objects in the array must match the quantity and datatype of the `DataColumn` objects in the `DataTable` object.

The `Load` method is used to insert data rows in the `DataTable` object. This method can also be used to update existing `DataRow` objects and load new objects. When calling the `Load` method, you need to pass an array of `DataRow` objects and a `LoadOption` enumeration value as parameters. These enumeration values could be:

- **PreserveCurrentValues:** This is the default value, which overwrites the original `DataRowVersion` without modifying the current `DataRowVersion`.
- **OverwriteRow:** Overwrites the original and the current `DataRowVersion` objects and changes the value of the `RowState` property to `Unchanged`.
- **UpdateCurrentValues:** Overwrites the current `DataRowVersion`, but does not modify the original `DataRowVersion`.

You can insert rows in a table in three ways:

- By providing the value for each column, and then, adding the row using the `Rows.Add()` method
- By inserting the entire row in a single line by passing comma-separated values
- By using the `LoadDataRow` method

Suppose you want to add data to the Vendors table, you need to first create the data row, and then, add data as shown in the following code:

```

private void GenerateDataTable()
{
    DataTable dataTable = new DataTable("Product");
    dataTable.Columns.Add(new DataColumn("Product_ID", typeof(int)));
    dataTable.Columns.Add(new DataColumn("Product_Name", typeof(string)));
    dataTable.Columns.Add(new DataColumn("Product_Price", typeof(decimal)));
    DataRow dataRow = dataTable.NewRow();
    dataRow["Product_ID"] = 1;
    dataRow["Product_Name"] = "Medicine1";
    dataRow["Product_Price"] = 100;
    dataTable.Rows.Add(dataRow);

    // Adding rows to the table in a single row
    dataTable.Rows.Add(2, "Medicine2", 120);
    dataTable.Rows.Add(3, "Medicine3", 50);
    dataTable.Rows.Add(4, "Medicine4", 25);
    // Load method will overwrite the existing data if the primary
key matches
    dataTable.LoadDataRow(new object[] { 4, "Medicine5", 45 },
LoadOption.OverwriteChanges);

    GridView_DataTable.DataSource = dataTable;
    GridView_DataTable.DataBind();
}
protected void Button_DataTable_Click(object sender, EventArgs e)
{
    GenerateDataTable();
}

```

In this example, all the insertion approaches previously discussed were used to insert rows into the table. Note that you are inserting a duplicate record for the product with ID Product_ID 4. However, this insertion will execute without any error because you have not yet set the primary key constraint. As an exercise, you can try the same example by assigning the primary key to Product_ID after you learn how to set the constraints.

Manipulating Data

Using DataAdapters, you can manipulate data. You can identify versions of data by reading the RowState. During data modification, you can also accept or reject the data. In addition, you can make duplicate data by cloning or copying it.

IDENTIFYING VERSIONS OF DATA

All data that undergoes manipulations start generating several additional versions. Similarly, when data is manipulated using DataAdapters, data in a DataRow object are maintained as three different versions—Original, Current, and Proposed. The first version of data that exists at the time of creation is **Current**. However, when this version is modified using the **BeginEdit** method, the new version is now called **Proposed** data. Subsequently, when the **EndEdit** method is called, the **Proposed** data gets reassigned as **Current**, and the **Current** data becomes **Original**, which denotes that it is an earlier version.

The version of the data stored in a DataRow object is stored using the **DataRowVersion** property. This property holds four predefined values namely **Current**, **Original**, **Proposed**, and **Default**.

Before attempting to retrieve a specific version of data, you can also query a DataRow object for the existence of that particular data row version. You can do this with the help of the

HasVersion method. The following code snippet demonstrates how to retrieve a string using RowState and DataRowVersion:

```
protected void btnDataRowVersion_Click(object sender, EventArgs e)
{
    DataTable dataTable = new DataTable("Product");
    dataTable.Columns.Add(new DataColumn("Product_ID",
    typeof(int)));
    dataTable.Columns.Add(new DataColumn("Product_Name",
    typeof(string)));
    dataTable.Columns.Add(new DataColumn("Product_Price",
    typeof(decimal)));
    DataRow dataRow = dataTable.NewRow();
    dataRow["Product_ID"] = 1;
    dataRow["Product_Name"] = "Medicine1";
    dataRow["Product_Price"] = 100;
    dataTable.Rows.Add(dataRow);

    // Adding row to the table in a single row
    dataTable.Rows.Add(2, "Medicine2", 120);

    Label lb1 = new Label();
    lb1.Text = string.Format("RowState: {0}<br />", dataRow.
    RowState);
    this.Controls.Add(lb1);
    foreach (string versionString in
    Enum.GetNames(typeof(DataRowVersion)))
    {
        DataRowVersion version = (DataRowVersion)Enum.Parse(typeof(
        DataRowVersion), versionString);
        if (dataRow.HasVersion(version))
        {
            Label lb2 = new Label();
            lb2.Text = string.Format("Version: {0} Value: {1}<br />",
            version, dataRow["Product_ID", version]);
            this.Controls.Add(lb2);
        }
        else
        {
            Label lb3 = new Label();
            lb3.Text = string.Format("Version: {0} does not
            exist.<br />", version);
            this.Controls.Add(lb3);
        }
    }
}
```

In this code snippet, you have added a new data row. Therefore, when you execute the code, a label is displayed with the message “Added” indicating that a new version has been added.

MANAGING DATA MODIFICATION

Usually, the changes made to data rows are accepted or rejected only after verifying their accuracy. This is done with the help of the AcceptChanges method, which can be used with the `DataRow`, `DataTable`, and `DataSet` objects. The function of the `AcceptChanges` method is to reset the `RowState` property to `UnChanged` and set the current row value to the original value, so that you can accept the changes after verifying the accuracy of the changed data. The `AcceptChanges` method also clears out the

RowError information and sets the HasError property to False. If you modify the DataRow objects, their RowState property changes to Modified. When you are ready to save the data, you can easily query the DataTable object for its changes by using the GetChanges method on the DataTable object, which returns a DataTable object that is populated only with the DataRow objects that have changed since the last time the AcceptChanges method was executed.

The following code shows you how to use the AcceptChanges and RejectChanges methods:

```
protected void AcceptChanges_Click(object sender, EventArgs e)
{
    DataTable dataTable = new DataTable("Product");
    dataTable.Columns.Add(new DataColumn("Product_ID",
    typeof(int)));
    dataTable.Columns.Add(new DataColumn("Product_Name",
    typeof(string)));
    dataTable.Columns.Add(new DataColumn("Product_Price",
    typeof(decimal)));
    DataRow dataRow = dataTable.NewRow();
    // Adding rows to the table in a single row
    dataTable.Rows.Add(1, "Medicine1", 100);
    ////AcceptChanges
    dataTable.AcceptChanges();
    Label lb1 = new Label();
    lb1.Text = "Accept Changes:" + dataRow.RowState +
    dataRow["Product_Price"] + "<br />";
    ////Modify the row
    dataRow["Product_Price"] = 200;
    Label lb2 = new Label();
    lb2.Text = "Modified the row:" + dataRow.RowState +
    dataRow["Product_Price"] + "<br />";
    // AcceptChanges
    dataTable.RejectChanges();
    Label lb3 = new Label();
    lb3.Text = "Rejected the changes:" + dataRow.RowState +
    dataRow["Product_Price"] + "<br />";
    // Delete the row
    dataRow.Delete();
    Label lb4 = new Label();
    lb4.Text = "Modified the row: " + dataRow.RowState + "<br />";
    this.Controls.Add(lb1);
    this.Controls.Add(lb2);
    this.Controls.Add(lb3);
    this.Controls.Add(lb4);
}
```

DUPLICATING DATA

There are some common operations that you can perform on a DataTable object such as creating a copy of the object, cloning the object, and importing rows into the object. To replicate a DataTable object, you can use the Copy and Clone methods. The Copy method copies the schema and the data of the object. The syntax for the Copy method is given here:

```
DataTable copy = vendor.Copy();
```

If you want only the copy of the `DataTable` schema without the data, you can use the `Clone` method. This method creates a copy of the empty `DataTable` object only with the schema. The syntax of this method is given next:

```
DataTable clone = vendor.Clone();
```

You can import a `DataRow` object into a `DataTable` using the `ImportRow` method. This operation can be performed only if the schema of the source and the destination `DataTable` objects are the same.

The following code shows you how to clone the `DataTable` object, `vendor`, and import a `DataRow` object into it:

```
DataTable clone = vendor.Clone();
clone.ImportRow(vendor.Rows[0]);
```

TAKE NOTE *

If you attempt to import a `DataRow` object that has a primary key value and whose value already exists in the `DataTable` object, a `ConstraintException` is thrown.

Using the `DataSet` Object

A `DataSet` object is a type of disconnected data object, whose data is stored in form of tables in memory.

A `DataSet` object is primarily used to represent the relational structure of tables in memory. It contains a collection of `DataTables` and `DataRelation` objects. It can also hold the primary key and *foreign key* constraints applied on the tables.

There are two ways to create a `DataSet` object: create it programmatically, or use an XML schema definition as the base for the tables-relationship structure. The following code example illustrates how to programmatically create a `DataSet` object that maintains the records of a drugstore:

```
private void GenerateDataSet()
{
    DataSet dataset = new DataSet();
    DataTable dataTableV = new DataTable("Product");
    dataTableV.Columns.Add(new DataColumn("Product_ID",
    typeof(Guid)));
    dataTableV.PrimaryKey = new DataColumn[] { dataTableV.
    Columns["Product_ID"] };

    dataTableV.Columns.Add(new DataColumn("Product_Name",
    typeof(string)));
    dataTableV.Columns.Add(new DataColumn("Product_Price",
    typeof(decimal)));

    DataTable dataTableP = new DataTable("Vendor");
    dataTableP.Columns.Add(new DataColumn("VendorID",
    typeof(Guid)));
    dataTableP.PrimaryKey = new DataColumn[] { dataTableP.
    Columns["VendorID"] };
    dataTableP.Columns.Add(new DataColumn("VendorName",
    typeof(string)));
    dataTableP.Columns.Add(new DataColumn("company",
    typeof(decimal)));

    dataset.Relations.Add("Product_Vendor", dataTableP.
    Columns["Product_ID"], dataTableV.Columns["VendorID"]);
}
```

In this code snippet, two tables—Product and Vendor—are created and a relationship between them is defined.

The following code snippet shows you how to populate data in the Product and Vendor tables:

```
DataRow dataRow = dataTableP.NewRow();
dataRow["Product_ID"] = 1;
dataRow["Product_Name"] = "Medicine1";
dataRow["Product_Price"] = 100;
dataTableP.Rows.Add(dataRow);

// Adding rows to the table in a single row
dataTableP.Rows.Add(2, "Medicine2", 120);
dataTableP.Rows.Add(3, "Medicine3", 50);
dataTableP.Rows.Add(4, "Medicine4", 25);
```

In this code snippet, a new globally unique identifier (GUID) is used to populate the VendorId column, which acts as the primary key for the Vendor table. When a new record is added to the vendor DataTable object, the data for the other columns are inserted. The DataTable objects of the Vendor and Product tables are associated using a DataRelation object called `Vendor_Product`.

Any `DataTable` object can be accessed by using the table name after the `DataSet` is created. For example, you can retrieve data from the Vendor or Product data tables as shown:

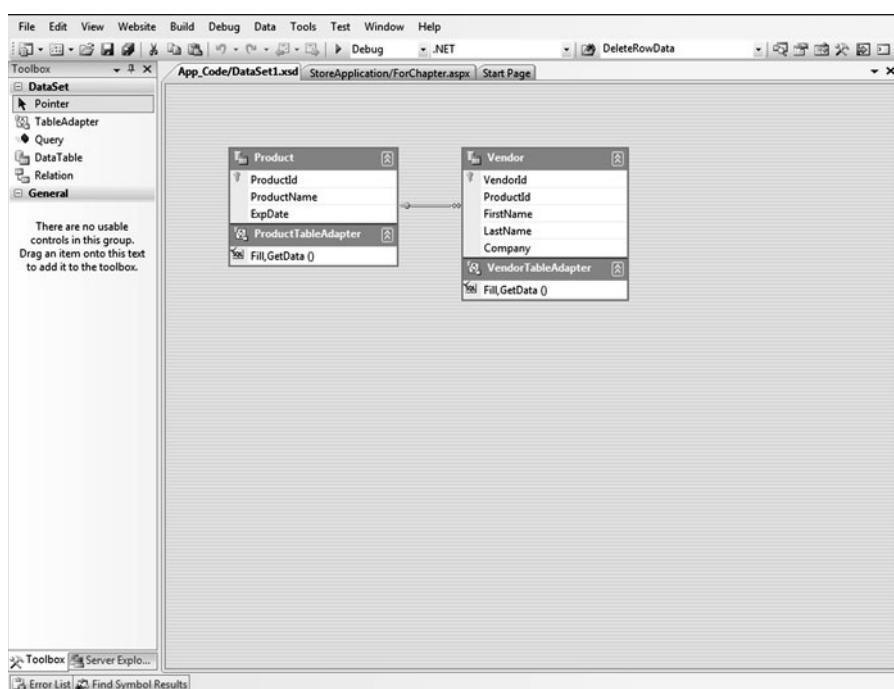
```
DataTable dtvendor = dsDrugStore.Tables["Vendor"];
```

If this code is executed with a misspelled table name, a runtime exception will occur. This error cannot be identified during compilation because the connection between the two tables is established only after the code is compiled and executed. Therefore, in order to catch the error at compilation, you can create a specialized `DataSet` class that will inherit from the base `DataSet` class and contain a property for each table.

A typed `DataSet` class can be created using XSD files, which can be modified with the help of the `DataSet` Editor. The `DataSet` Editor provides a graphical user interface (GUI) to create and modify XSD files. Figure 4-3 shows the `CompanyList` `DataSet` that is loaded into the `DataSet` Editor.

Figure 4-3

`DataSet` editor with a `DataSet` loaded



■ Manipulating Data Using DataSet, DataAdapter, and DataReader Objects



Data manipulation is required by all data-driven applications. Data stored in data sources may not make enough sense to the end user if displayed as is. For example, in an employee management system, the address of an employee may include information like street address, state, country, and zip code. This information may be stored in different fields. At the same time, instead of country and state names, the table may store theCountryID and StateID. Displaying the state and country ID to the end user may not be appropriate. For this reason, data manipulation is required at the time of data insertion or retrieval.

When manipulating data, you often use both connected and disconnected objects. To read data, you need to use the `DataReader` object, and to manipulate or update data, you need to use the `DataSet` object. You also use the `DataAdapter` object for such operations.

Retrieving Data Using DataReader

To retrieve data from a database, you create an instance of the `DataReader` object.

To create an instance of the `DataReader` object, use the `ExecuteReader` method of the `command` object, as shown in the following code:

```
OleDbDataReader reader = command.ExecuteReader();
```

In this code, `reader` is a `DataReader` object and `command` represents a valid `command` object, which usually contains a query to the database. The `ExecuteReader` method executes this query and retrieves the resultant rows and columns into the `DataReader` object, `reader`. Now, you can use the `Read` method of the `DataReader` object to read every row or column retrieved. You can also use the typed accessor methods (`GetDataTime`, `GetGuid`, `GetInt32`, and `GetDouble`) to convert column values into display-friendly datatypes.

The following code example shows you how to use the `DataReader` object to read data from a table:

```
private void ReadOrderData(string connectionString)
{
    string queryString;
    queryString = "select * from ProductsOrder;";
    using (OleDbConnection connection = new
    OleDbConnection(connectionString))
    {
        OleDbCommand command = new OleDbCommand(queryString,
        connection);
        connection.Open();
        OleDbDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Response.Write(String.Format("{0}, {1}",
            reader.GetOrdinal("Product_ID"), (reader[1]).ToString()));
        }
        reader.Close();
    }
}
```

This code retrieves selected columns from a table. Similarly, to obtain multiple resultsets, you can use the `NextResult` method of the `DataReader` object and iterate through the

resultsets in order. Sometimes, a stored procedure may return more than one record set. The following code example shows you how to process the results of two SELECT statements using the `NextResult` method:

```
private void ReadOrderData(string connectionString)
{
    string queryString;

    queryString = "select * from ProductsOrder;" + "Select * from
Product";
    using (OleDbConnection connection = new
OleDbConnection(connectionString))
    {
        OleDbCommand command = new OleDbCommand(queryString,
connection);
        connection.Open();
        OleDbDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Response.Write(String.Format("{0}, {1}",
reader.GetOrdinal("Product_ID"), (reader[1]).ToString()));
        }
        reader.Close();
    }
}
```

CLOSING THE DAREADER

After using a `DataReader` object to read data from a database, it is mandatory that you close the object. If you do not close it, the connection will be in use, and you will not be allowed to execute any more commands using that connection. In other words, you cannot create another `DataReader` object without closing the existing one. To close the `DataReader`, use the following code:

```
reader.Close();
```

Working with the DataAdapter Object

In ADO.NET, a `DataAdapter` object acts as a bridge between a data source and a `DataSet` object and facilitates two-way communication between them.

The primary function of the `DataAdapter` object is to populate the `DataSet` object that has been created and stored in memory with data from the data source. Using the different properties of the `DataAdapter` object, you can manipulate the data in the `DataSet` object and update it back to the data source. The following Properties of `DataAdapter` are used to manipulate data in a data source:

- **SelectCommand:** Retrieves all the rows or a specific set from a table in the data source.
- **InsertCommand:** Inserts new rows of data into the data source.
- **UpdateCommand:** Updates existing rows in the data source.
- **DeleteCommand:** Deletes specified rows from the data source.

All the `DataAdapter` objects (represented programmatically as `SqlDataAdapter`) implement the `IDbDataAdapter` interface, and are inherited from a base class, `dbDataAdapter`.

In addition to inherent properties that help data manipulation, the `DataAdapter` object also provides various methods for the same purpose. These methods are listed in Table 4-6.

Table 4-6

Methods of the DataAdapter object

METHOD	DESCRIPTION
Fill	Populates the table with rows retrieved from the data source within the memory.
FillSchema	Configures the in-memory table schema with the schema in the data source.
GetFillParameters	Returns the parameters set in the query statement.
Update	Updates the data source tables with the latest data modified in the in-memory tables. Data manipulation is performed using the respective INSERT, UPDATE, or DELETE statements.

The `Fill` method is used to populate the `DataSet` object with the results of the query string set in the `SelectCommand` property of the `DataAdapter`. The `DataTable` object is passed as an argument for the `Fill` method. To return the column names and their datatypes that will be used in the `DataSet` object, the `Fill` method uses a `DataReader` object.

If the primary key exists on a table, the `Fill` method will overwrite data in the `DataSet` object with the data in the data source, but only for rows where the primary key column values match the rows returned from the data source. In the absence of the primary key, the data is appended to the tables in the `DataSet` object.

The following code example illustrates how to use an `SqlDataAdapter` object to populate data from a DrugStore database:

```
private static DataTable ReadDisconnectedData(string connectionString)
{
    DataSet dataSet;
    string queryString;

    dataSet = new DataSet();
    queryString = "select * from Products;";

    using (OleDbConnection connection = new
OleDbConnection(connectionString))
    {
        OleDbDataAdapter adapter = new
OleDbDataAdapter(queryString, connection);
        adapter.Fill(dataSet, "PersonalInfo");
    }
    return dataSet.Tables[0];
}
```

In this example, the `DataAdapter` is used to fill the `DataSet` with the data from the database for the `Product` table.

On query execution, if a `DataAdapter` encounters multiple resultsets, then that many tables are created in the `DataSet`. These tables are named with an incremental default name `TableN`. Therefore, the first table containing the first resultset will be named `Table0` and so on.

POPULATING A DATASET FROM MULTIPLE DATAADAPTERS

There is no restriction on the number of `DataAdapter` objects that can be associated with a `DataSet` object. Each `DataAdapter` can be used to fill one or more `DataTable` objects. `DataRelation` and `Constraint` objects can also be added to the `DataSet` locally, which allows you to relate data from different data sources, such as Microsoft SQL Server and DB2.

The following code example illustrates how to populate a `DataSet` with a list of customers from the DrugStore database stored in Microsoft SQL Server 2005 and the Vendor database

stored in Microsoft Access 2007. The retrieved tables are related using a `DataRelation` object. The list of customers is then displayed based on the orders they place:

```
protected void btnDataAdapter_Click(object sender, EventArgs e)
{
    string Sq1DBConn;
    Sq1DBConn = ConfigurationManager.ConnectionStrings
    ["SQLWebConnectionString"].ConnectionString;
    DataSet dataSet;
    string queryString;
    dataSet = new DataSet();
    queryString = "select * from Products;";
    using (SqlConnection connection = new SqlConnection(Sq1DBConn))
    {
        Sq1DataAdapter adapter = new
        Sq1DataAdapter(queryString, connection);
        adapter.Fill(dataSet, "Product");
    }
    queryString = "select * from Vendor;";
    using (OleDbConnection connection = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;data Source=DrugStore.mdb;"))
    {
        OleDbDataAdapter adapter = new
        OleDbDataAdapter(queryString, connection);
        adapter.Fill(dataSet, "Vendor");
    }
    DataRelation relation = dataSet.Relations.
    Add("VendProd", dataSet.Tables["Vendor"].Columns["VendorID"], dataSet.
    Tables["Product"].Columns["ProductID"]);
    GridView1.DataSource = dataSet;
    GridView1.DataBind();
}
```

USING PARAMETERS WITH DATAADAPTER

As already mentioned, a `DataAdapter` object has four properties that help data manipulation, namely, `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand`. The `SelectCommand` property returns data from the data source and must be set before calling the `Fill` method. The `InsertCommand`, `UpdateCommand`, and `DeleteCommand` properties must be set before calling the `Update` method.

For example, if you want to add a set of rows to the data source, you need to first assign the `INSERT` statement to the `InsertCommand` property, and then, call the `Update` method. When executing the `Update` method, the `DataAdapter` uses the respective `Command` property and passes current information about the newly inserted rows using a `Parameters` collection.

The following code example demonstrates the use of these `DataAdapter` properties:

```
// Code for deleting the record using OleDbDataAdapter through
parameters.
private OleDbDataAdapter DeleteRowData(string connectionString)
{
    string queryString;
    queryString = "Delete from Products where Product_ID =
@Product_ID;";
    using (OleDbConnection connection = new
    OleDbConnection(connectionString))
    {
```

```
connection.Open();
OleDbDataAdapter adapter = new OleDbDataAdapter();
adapter.SelectCommand = new OleDbCommand(queryString, connection);
        adapter.DeleteCommand.Parameters.Add
        ("@Product_ID", OleDbType.Integer, 5, "Product_ID");
    return adapter;
}
// Code to update the record using OleDbDataAdapter and parameters.
private OleDbDataAdapter UpdateRowData(string connectionString)
{
    string queryString;
    int count;
    queryString = "Update Product SET ProductPrise =
@Product_Price where Product_ID = @Product_ID;";
    using (OleDbConnection connection = new
OleDbConnection(connectionString))
    {
        connection.Open();
        OleDbDataAdapter adapter = new OleDbDataAdapter();
        adapter.SelectCommand = new
OleDbCommand(queryString, connection);
        adapter.UpdateCommand.Parameters.Add
        ("@Product_ID", OleDbType.Integer, 5, "Product_ID");
        adapter.UpdateCommand.Parameters.Add ("@Product_Price",
OleDbType.Integer, 5, "Product_Price");
        return adapter;
    }
}
```

In this code example, note that an `SqlDataAdapter` object has been created and its `MissingSchemaAction` property has been set to `AddWithKey` so that additional schema information can be retrieved from the database. In this code, `SourceColumn` is the name of the `DataColumn` from the `DataRow` where the value of the `Parameter` will be retrieved, and `SourceVersion` specifies the `DataRowVersion` that the `DataAdapter` uses to retrieve the value. You can pass `SourceColumn` and `SourceVersion` as arguments to the `Parameter` constructor, or set them as properties of an existing `Parameter`.

ADO.NET helps you connect to different types of data sources with the help of different providers, and each of these providers has its own definition of the `DataAdapter` class. One such definition that SQL Server provides is `SqlClient.Parameters`. For example, to define a parameter for the `UpdateCommand` in `SqlClient`, you can use the following code:

```
@VendorID(set VendorID=@VendorID) and
@OldVendorID(where VendorID = @OldVendorID)
```

The `@VendorID` parameter will update the `VendorID` column to the current value in the `DataRow` object. This will result in a `VendorID` `SourceColumn` with a `SourceVersion` of `Current` being used. The `@OldVendorID` will identify the current row in the data source only because the matching column value is found in the original version of the row.

ADDING EXISTING CONSTRAINTS INTO A DATASET

To ensure the presence of primary key constraints in the `DataTable` objects within a `DataSet`, before filling the `DataTable` objects with data, you need to add the schema information to the `DataSet`. Remember that the `Fill` method of the `DataAdapter` object

fills the **DataSet** only with data from the rows and columns of the table in the data source. The **Fill** method does not consider any of the constraints in the data source table when filling data. Therefore, you need to use the **FillSchema** method and populate the **DataSet** with existing primary key constraints. Alternatively, before calling the **Fill** method, if you set the **MissingSchemaAction** property of the **DataAdapter** to **AddWithKey**, all the primary key constraints are also copied along with the data. Note that neither of these methods copy the foreign key constraints. They should be created manually after filling the **DataTable** objects.

The primary advantage of creating the schema in the **DataSet** is that, after populating the **DataSet** tables for the first time, if you make any subsequent update **Fill** calls, the new rows are matched against the existing rows and the data is filled. In the absence of the schema information, the new rows are simply appended to the **DataSet**, thus duplicating the records in your dataset.

TAKE NOTE*

Using the **FillSchema** method or the **MissingSchemaAction** property may affect the performance of the application because this method and property require additional processing. Therefore, it is good practice to verify whether the data source has pre-existing primary key constraints before deciding whether to use it.

The following lines of code show how to add schema information to a **DataSet** using the **FillSchema** method:

```
DataSet dsDrgStr = new DataSet();
custAdapter.FillSchema(dsDrgStr, SchemaType.Source, "Vendor");
custAdapter.Fill(dsDrgStr, "Vendor");
```

The following lines of code show how to add schema information to a **DataSet** using the **MissingSchemaAction** property:

```
DataSet custDataSet = new DataSet();
custAdapter.MissingSchemaAction = MissingSchemaAction.AddWithKey;
custAdapter.Fill(custDataSet, "Vendor");
```

TAKE NOTE*

You know that auto-increment columns are those columns in a table that automatically increment its row value every time a row is inserted. This value will usually be a unique identifier for the row. If a data source table contains an auto-increment column, the **DataSet** can be filled by using any of the following techniques:

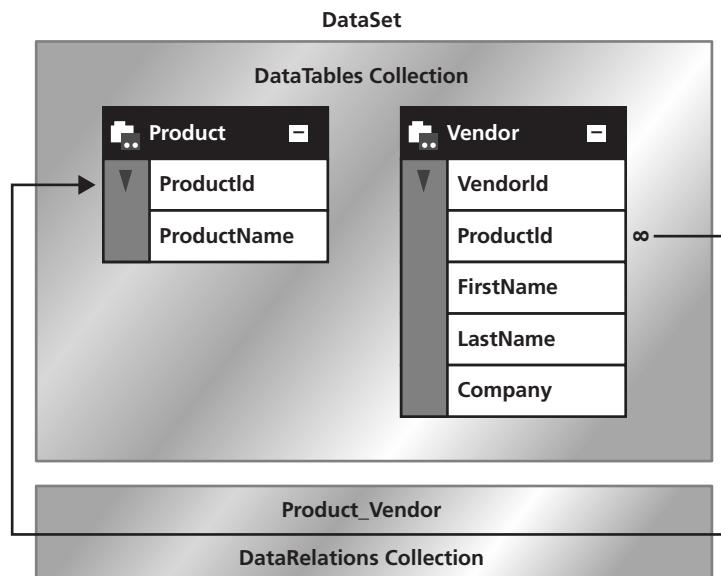
- Letting a stored procedure return the auto-increment values and map it to the **DataTable** column.
- Making a stored procedure return the auto-increment values as the first row in the resultset.
- Executing an additional select statement explicitly by using the **RowUpdated** event of the **DataAdapter**.

MAPPING DATATABLE AND DATACOLUMN OBJECTS USING DATAADAPTER

To read data from a **DataSet**, the adapter must first find the table from the **TableMappings** collection. The tables in this collection are named using the mechanism of incremental default naming. If the table that the adapter is looking for is found in the collection, the **DataAdapter** reads the table name and then locates the **DataTable** object in the **DataSet** having a similar name as specified in the mapping. If the table name is not found in the collection, a new table is created and is filled with data from the retrieved **DataSet**. Figure 4-4 illustrates this mapping mechanism.

Figure 4-4

Mapping a table with the TableMappings collection



If you map the **Vendor** table using **DataAdapter**, a table named **Vendor** will be present in the **DataSet**. If the table name is not found, a new table named **Vendor** will be created and filled in the **DataSet**. If the table named **Vendor** is found in the **DataSet**, the data will be simply merged with the **DataSet**. All together, **TableMapping** is a collection of objects of type **DataTableMappingCollection**, and each **DataTableMapping** object in the collection defines a pair of names, which consist of the source name and the in-memory table name.

The following code example shows you how to configure table mapping:

```
DataSet dataset = new DataSet();
DataTableMapping dtmVendor, dtmProduct, dtmProductOrder;
dtmVendor = adapter.TableMappings.Add("Table", "Vendor");
dtmProduct = adapter.TableMappings.Add("Table1", "Product");
dtmProductOrder = adapter.TableMappings.Add("Table2", "ProductOrder");
adapter.Fill(dataset);
```

The default name to be mapped against the names that you provided must coincide with the default names originated in the resultset using the **Fill** method.

Now, let us look at an example of table mapping. Create a **DataTableMapping** with the name **Customers** and a **DataTable** with the name **DrugStoreSchema**. Then, map the rows returned by the **SELECT** statement to the **DrugStoreSchema** **DataTable**, as shown in the following code example:

```
using (OleDbConnection connection = new OleDbConnection
(connectionString))
{
    OleDbDataAdapter adapter = new
OleDbDataAdapter(queryString, connection);
    ITableMapping tmVendor =
adapter.TableMappings.Add("Vendor", "DrugStoreSchema");
    tmVendor.ColumnMappings.Add("VendorID", "ClientID");
    tmVendor.ColumnMappings.Add("FirstName", "ClientName");
    tmVendor.ColumnMappings.Add("ContactNo", "ClientContact");
    adapter.Fill(dataSet, "Vendor");
}
```

In this example, if the **SelectCommand** property returns multiple tables, the **Fill** method automatically generates table names with incremental values for the tables in the **DataSet**,

starting with the specified table name and continuing in the form TableNameN. You can use table mappings to map the automatically generated table name to a name that you want. For example, for a `SelectCommand` that returns two tables, Customers and Orders, you need to call the `Fill` method as given:

```
adapter.Fill(ProductDataSet, "Products")
```

This code creates two tables, ProductDataSet and ProductOrder. Through mapping you can ensure that the other table is named Order instead of ProductOrder by mapping the source table of ProductOrder to the `DataSet` Order as shown in the following code:

```
adapter.TableMappings.Add("ProductOrder", "Orders")
adapter.Fill(ProductDataSet, "Product");
```

USING PAGING TO DISPLAY RESULT SETS

To break the query results into smaller chunks of manageable data, you can adopt a paging approach. ASP.NET allows you to apply paging with the help of the `DataAdapter` class `Fill` method. When using this method, be sure to set the `startRecord` parameter to the point where paging should start and the `maxRecords` parameter to the maximum number of records allowed in a page.

The following code example shows you how to use the `Fill` method to page query results on the basis of five records per page:

```
protected void btnDataAdapter_Click(object sender, EventArgs e)
{
    string SqldbConn;
    SqldbConn =
ConfigurationManager.ConnectionStrings["WebConnectionString"].
ConnectionString;
    string queryString;
    DataSet dataSet = new DataSet();
    queryString = "select * from ProductsOrder;";
    int Index = 0;
    int MaxCount = 10;
    using (SqlConnection connection = new SqlConnection
(SqldbConn))
    {
        SqlDataAdapter adapter = new
SqlDataAdapter(queryString, connection);
        // Code for paging
        adapter.Fill(dataSet, Index, MaxCount, "Product");
    }
    // Fill the GridView1 with DataSet
    GridView1.DataSource = dataSet;
    GridView1.DataBind();
}
```

In this example, the page size is set while filling the `DataSet`. However, the query string will return all the records from the `ProductOrder` table at one go. Therefore, to fill only the first ten records at a time, you can use the `TOP` and `ORDER BY` clauses in the query statement as:

```
"Select TOP" + MaxCount + "from ProductOrder ORDER BY Product_ID";
```

This code example retrieved the first set of records. To retrieve the next set, you need to preserve the unique identifier of the last row of the previous set and pass it to the command. In addition, you need to increment the `currentIndex` value by the page size, set it in the

`StartRecord` parameter, and then pass the page size to the `maxRecord` parameter as shown in the following lines of code:

```
string lastRecord = dataSet.Tables["Vendor"].Rows[pageSize - 1]
["VendorID"].ToString();
currentIndex += pageSize;

dataSet.Tables["Vendor"].Rows.Clear();

adapter.Fill(dataSet, currentIndex, pageSize, "Vendor");
```

UPDATING DATA SOURCES WITH DATAADAPTER

Similar to the `Fill` method, the `Update` method of the `DataAdapter` object passes the `DataSet` object and the `DataTable` or the `DataTable` name (optional) as parameters. The `Update` method updates the data source with the `DataSet` that contains the changes. If the `DataTable` is passed along with the `DataSet`, the `DataTable` helps identify the table in which the changes are made. In the absence of the `DataTable`, the first table of the `DataSet` is selected.

On any kind of update to the data source, the `DataAdapter` object analyzes the changes made and executes the appropriate command, such as `INSERT`, `UPDATE`, or `DELETE`. Suppose you insert a new record, change an existing record at the same time, or delete a record from a table. In this case, the `DataAdapter` analyzes the changes made to the `DataTable` and executes the appropriate commands.

The `DbCommandBuilder` object is used to automatically build the `DeleteCommand`, `InserCommand`, and the `UpdateCommand` properties for a `DataAdapter`. However, this is beneficial only when the `DataTable` maps to a single database table.

MAPPING DATASET VALUES USING UPDATEROWSOURCE

After an `UDATE` action has taken place, the data retrieved from the data source needs to be mapped back to the `DataTable`. You can do this by using the `UpdatedRowSource` property of the `DbCommand` object. You can control the output parameter returned by the `DataAdapter` command by setting the `UpdatedRowSource` property to one of the `UpdateRowSource` enumeration values. The `set` property determines whether the changed row in the `DataTable` is ignored or applied. Table 4-7 lists the `UpdateRowSource` enumeration values that affect the behavior of a `DataAdapter`.

Table 4-7

UpdateRowSource
Enumeration Values

ENUMERATION	DESCRIPTION
Both	The output parameter as well as the first row obtained from the resultset may be mapped to the updated row in the dataset.
FirstReturnedRecord	The first row obtained from the resultset alone is mapped to the updated row in the dataset.
None	The output parameter or rows obtained from the resultset are ignored.
OutputParameters	The output parameters obtained from the resultset alone is mapped to the updated row in the dataset.

Conflicts between the `DataSet` and the data source are not restricted to the changes you made. A conflict could be due to the changes made by other connected clients. If the data filled last in your `DataSet` does not match the current data in the data source, then the `Update` method can be used to resolve the conflict by updating the data source. There are two ways to handle the exception that occurs when calling the `Update` method. You can use the `RowUpdate` event to respond to the error that occurred, or set the `DataAdapter.ContinueupdateOnError` to `True` prior to calling the `Update` method. The error is stored in the `RowError` property specific to the row, and you can respond to the error stored there after the update is complete.

It is very important to remember that by calling the `AcceptChanges` method, the `DataSet`, `DataTable`, or `DataRow` will overwrite all the original values with the current value. Also, if the unique identifier field value of the row is modified after calling `AcceptChanges`, then the original value will no longer match the value in the data source. When the `Update` method of the `DataAdapter` is called, the `AcceptChanges` method is called automatically. The original value can be preserved by creating an event handler for the `RowUpdate` event and setting the `Status` property to `SkipCurrentRow`. You can also set the `AcceptChangesduringUpdate` property of the `DataAdapter` to `False` when calling the `Update` method.

The following code example will help you understand how to update rows and set the `UpdateCommand` property:

```
protected void btnAdapterUpdate_Click(object sender, EventArgs e)
{
    string DBConn = ConfigurationManager.
        ConnectionStrings["WebConnectionString"].ConnectionString;
    string queryString;
    int count;
    queryString = "Update Product SET ProductPrise =
@Product_Price where Product_ID = @Product_ID;";
    using (OleDbConnection connection = new
        OleDbConnection(DBConn))
    {
        OleDbDataAdapter adapter = new OleDbDataAdapter();
        // Select command
        queryString = "Select * from Vendor table";
        adapter.SelectCommand= new
        OleDbCommand(queryString, connection);
        // Update command
        queryString = "Update Vendor set ContactNo =
@NewContactNo where VendorID = @VendorID";
        adapter.SelectCommand= new
        OleDbCommand(queryString,connection);
        // Raise events
        adapter.RowUpdated += new
        OleDbRowUpdatedEventHandler( OnRowUpdated );
        adapter.RowUpdating += new
        OleDbRowUpdatedEventHandler( OnRowUpdating );
        adapter.FillError += new
        FillErrorEventHandler( OnFillError );
        // Add parameters
        OleDbParameter param1 = adapter.UpdateCommand.Parameters.
        Add("@NewContactNo", OleDbType.NVarChar, 10, "NewContactNo" );
        OleDbParameter param2 = adapter.UpdateCommand.Parameters.
        Add("@VendorID", OleDbType.Int,5,"VendorID" );
        param2.SourceVersion = DataRowVersion.Original;
        connection.Open();
        DataSet dataset = new DataSet();
        adapter.Fill(dataset);
        // Changing the DataSet row value
        dataset.Tables[0].Rows[0]["ContactNo"] = "09562345223";
        adapter.Update(dataset);
        // Remove events
        adapter.RowUpdated -= new
        OleDbRowUpdatedEventHandler( OnRowUpdated );
    }
}
```

```
        adapter.RowUpdating -= new  
OleDbRowUpdatedEventHandler( OnRowUpdating );  
  
        connection.Close();  
    }  
}  
// Method to handle OnRowUpdating  
private void OnRowUpdating(object sender, SqlRowUpdatingEventArgs args)  
{  
    if (args.StatementType == StatementType.Delete)  
    {  
        label lbUpdating = new label();  
        lbUpdating.text = args.Row["VendorID",  
DataRowVersion.Original];  
        this.Controls.Add(lb1);  
    }  
}  
  
// Method to handle OnRowUpdated  
private void OnRowUpdated(object sender, SqlRowUpdatedEventArgs args)  
{  
    if (args.Status == UpdateStatus.ErrorsOccurred)  
    {  
        label lbUpdated = new label();  
        args.Row.RowError = args.Errors.Message;  
        args.Status = UpdateStatus.SkipCurrentRow;  
        lbUpdated.text = args.Row["VendorID",  
DataRowVersion.Current];  
    }  
}  
// Method to handle OnFillError  
private void OnFillError(object sender, FillErrorEventArgs args)  
{  
    if (args.Errors.GetType() == typeof(System.OverflowException))  
    {  
        DataRow objDataRow = args.DataTable.Rows.Add(new object[]  
{ args.Values[0], args.Values[1], DBNull.Value });  
        args.RowError = "Data OverflowException  
Encountered" + args.Values[2];  
        args.Continue = true;  
    }  
}
```

It is not common practice to sequence the **INSERT**, **UPDATE**, and **DELETE** operations on a table, but sometimes it becomes necessary. For example, if the **Product_ID** needs to be changed, the **Order** cannot be raised and the new order data cannot be stored in the **ProductOrder** table with the new **Product_ID** unless it is changed in the **Product** table. Here, the **Product_ID** is a primary key in the **Product** table. In such cases, the **INSERT**, **UPDATE**, and sometimes **DELETE** operations need to be carried out in a sequence. For example, if the address of a person is changed in the **Address** table that holds a primary key, a new record based on the changed address cannot be inserted in the **Dispatch Post** table unless the update in the **Address** table is fired. The order of execution is essential here because the **Address ID** acts a foreign key in the **Dispatch Post** table. In such situations, you can also choose to modify only the updated rows in your data source by selecting the records from the **DataTable** with the help of the **Selected** method. This method is triggered based on the **RowState** value, which returns the **DataRow** array to the **Update** method.

TAKE NOTE *

You can sequence the INSERT, UPDATE, and DELETE operations using the following code:

```

protected static void SeqQueryExec()
{
    string connectionString =
ConfigurationManager.ConnectionStrings["WebConnectionString"].
ConnectionString;
    string queryString = "select * from Product";
    using (OleDbConnection connection = new
OleDbConnection(connectionString))
    {
        OleDbDataAdapter adapter = new
OleDbDataAdapter(queryString, connection);
        adapter.Fill(dataset, "Product");
        DataTable table = dataset.Tables["Product"];

        // Update the record in the DataTable
        adapter.Update(table.Select(null, null,
DataViewRowState.ModifiedCurrent));
        Label lb1 = new Label();
        lb1.Text = "Record Updated in the DataTable";

        // Add new record in the DataTable
        adapter.Update(table.Select(null, null,
DataViewRowState.Added));
        Label lb2 = new Label();
        lb2.Text = "Record Inserted in the DataTable";
        this.Controls.Add(lb1);
        this.Controls.Add(lb2);
    }
}

```

From the code, you can see that first, the `DataTable` is updated, and then, a new record is inserted in the table.

Understanding DataAdapter Events

You should know about the various events that `DataAdapter` objects raise. You can use these events of the `DataAdapter` object to respond to the changes in the data source data. Table 4-8 describes the various events.

Table 4-8

Events of the `DataAdapter` object

EVENT	DESCRIPTION
RowUpdating	Indicates the beginning of an UPDATE, INSERT, or DELETE operation on a row by calling the <code>Update</code> method.
RowUpdated	Indicates the completion of an UPDATE, INSERT, or DELETE operation on a row by calling the <code>Update</code> method.
FillError	Indicates that an error has occurred during the <code>FILL</code> operation.

USING THE ROWUPDATING AND ROWUPDATED EVENTS

The `RowUpdating` and `RowUpdated` events are raised to modify the data source data with the `DataSet` data. These events are raised before and after the `Update` method call of the `DataAdapter`. The `RowUpdating` event provides additional handling to the behavior of the `Update` method and helps retain the reference to the updated row for later use, such as for canceling the current update and scheduling a batch process to run later. The `RowUpdated`

event is used to respond to the error that may arise while the update is being performed on the data source from a **DataSet**. You can add exception or error information to the **DataSet** as well as retry logic.

Along with the **RowUpdatingEventArgs** for the **RowUpdating** event and the **RowUpdatedEventArgs** for the **RowUpdated** event, various other arguments accompany these events, such as:

- **Command**: Holds the reference of the **Command** object in use while performing the **Update** method.
- **Row**: Holds the reference to the **DataRow** object that contains the update information.
- **StatementType**: Holds information about the type of update that is being performed.
- **TableMapping**: Maps the data of **DataSet** and the data source during update.
- **Status**: Checks the status of the operation. It also stores information, which helps identify whether any error has occurred during the operation.

Table 4-9 describes various values of the **Status** property that control action during the update.

Table 4-9

Values of the Status property

STATUS	DESCRIPTION
ErrorsOccurred	Indicates that an error has occurred and throws the exception and aborts the update.
SkipCurrentRow	If an error occurs on a particular row, then it ignores the current row and continues the update.
Continue	Continues the update. It also indicates that no error has occurred.
SkipAllRemainingRows	If an error occurs, then it stops and aborts the update without throwing an exception.

Let us see how to use the **RowUpdated** and **RowUpdating** events in a dataset.

The event is raised by writing the following code:

```
// Raise events
adapter.RowUpdated += new OleDbRowUpdatedEventHandler( OnRowUpdated );
adapter.RowUpdating += new OleDbRowUpdatedEventHandler( OnRowUpdating );
adapter.FillError += new FillErrorEventHandler( OnFillError );

// Similarly, you can remove the events using the following code:
// Remove events
adapter.RowUpdated -= new OleDbRowUpdatedEventHandler( OnRowUpdated );
adapter.RowUpdating -= new OleDbRowUpdatedEventHandler( OnRowUpdating );
```

USING THE FILLERROR EVENT

During the data load to the **DataSet** from a data source using the **Fill** method, there is a possibility that the row being added may not be converted to the .NET Framework type, or that it may be converted with loss of precision. For such instances, **DataAdapter** provides the **FillError** event to handle the error. The data row will not be added to the **DataTable** in case the error occurs during the **FILL** operation. You can ignore the excluded row and continue the **FILL** operation or resolve the error and consider adding the row.

Table 4-10 lists the properties that are passed as the **FillErrorEventArgs** to the **FillError** event.

Table 4-10

Arguments of the FillError event

CERTIFICATION READY?

Manipulate data by using DataSet and DataReader objects.

3.2

PROPERTY	DESCRIPTION
Errors	Provides information about the exception that occurred.
DataTable	Represents the DataTable object on which the Fill method is executed and on which the error occurred.
Values	Represents an array of the row value where the error occurred while filling the data. The array value ordinal is directly correlated to that of the columns in the rows being added. Value[0] will correlate to the first column of the row.
Continue	Allows you to decide whether you need to throw an exception. If Continue is set to True , then an exception will be raised while the Fill method is used on the DataSet . If Continue is set to False then the FILL operation will be stopped.

■ Using Data Source Controls



THE BOTTOM LINE

ADO.NET has made many controls to simplify and accelerate the development process available, such as the data source controls. Access to the data source using the data source controls is not restricted to the database. These controls allow access with XML files and middle-tier business objects as well. You can easily connect to the data source to retrieve data and bind with other controls without massive coding.

X REF

Accessing data with help of LINQ will be covered in detail in Lesson 5.

Using Built-In Data Source Controls

The .NET Framework provides several built-in data source controls. These data source controls support data binding with data sources for different data-binding scenarios.

USING LINQDATASOURCE CONTROL

This control helps retrieve data from data source tables or from in-memory data tables, with the least amount of coding. While fetching data from the SQL server, you can configure this control to handle **INSERT**, **UPDATE**, and **DELETE** operations.

USING SQLDATASOURCE CONTROL

This control uses SQL commands for exchanging and modifying data. The **sqlDataSource** control can be used with Microsoft SQL Server, Oracle, OLEDB, and ODBC databases. The **DataReader** and **DataSource** objects hold the resultant data from the **sqlDataSource** control. **DataSet** helps cache the data, and it also supports sorting and filtering.

USING ENTITYDATASOURCE CONTROL

This control is based on the Entity Data Model (EDM), which is used for object-relation mapping by the .NET Framework. In addition, this control is also used for ADO.NET Data Services. Data binding takes place with the help of Entity-SQL (eSQL) querying language. The **EntityDataSource** control also supports queries created by the **ObjectQuery(T)** class.

USING OBJECTDATASOURCE CONTROL

This control interacts with the business objects or classes, which are basically the middle tier of a web application. In the case of the **IEnumerable** interface, the retrieved data

is always returned as a **DataSet**, **DataTable**, or **DataView** object. The values from the bound controls are passed as parameters to the source objects when a method is called.

X REF

Accessing data with help of XML will be covered in detail in Lesson 5.

USING XMLDATASOURCE CONTROL

This control uses XML for reading and writing data. It can also use schema to expose data as typed members. You can easily create TreeView and Menu controls using the XML resultset of the XMLDataSource control. In order to restructure the raw data as an XML file, you can apply the XSLT transformation according to the control used. Filtering XML is made simple by the XPath expression. You can directly traverse to the nodes to return selected specific nodes that have specific values.

USING ACCESSDATASOURCE CONTROL

This control is specially designed to work with Microsoft Access database, that is, with the .mdb files. It works like the SqlDataSource control, which can be used to control the data fetching and modifying operations on the database.

USING SITEMAPDATASOURCE CONTROL

This control is meant to provide site navigation data using the ASP.NET site map. Site navigation can be customized using the site map, with web server controls like TreeView and with drop-down controls, which are not actually meant for site navigation. The Menu control is most commonly used for site navigation.



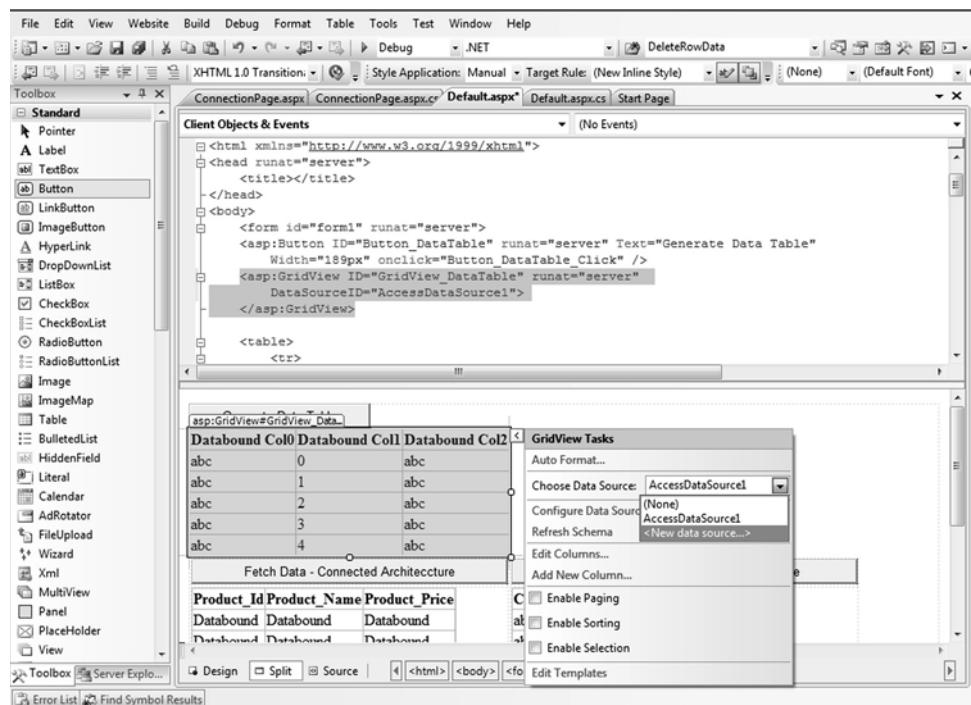
CREATE DATASOURCE CONTROL

To use the AccessDataSource control in your application:

1. Drag and drop a GridView control on the page.
2. Click the top-right icon of the GridView control. In the Choose Data Source dropdown list box, click <New data source . . .>. The Data Source Configuration Wizard opens, as shown in Figure 4-5.

Figure 4-5

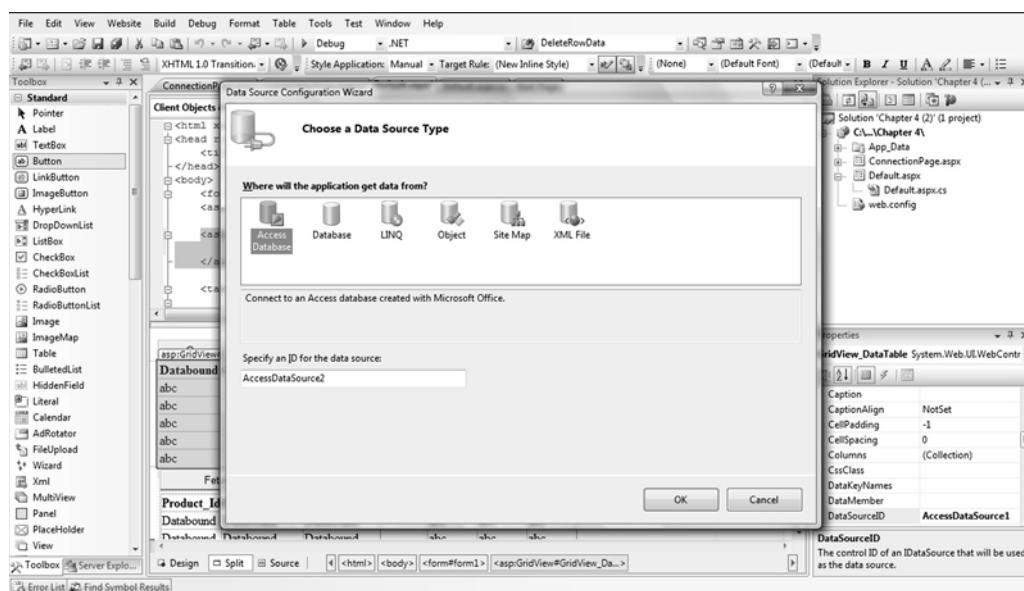
New data source option



3. Select Access Database as the database, and click OK. The Select Microsoft Database dialog box opens, as shown in Figure 4-6.

Figure 4-6

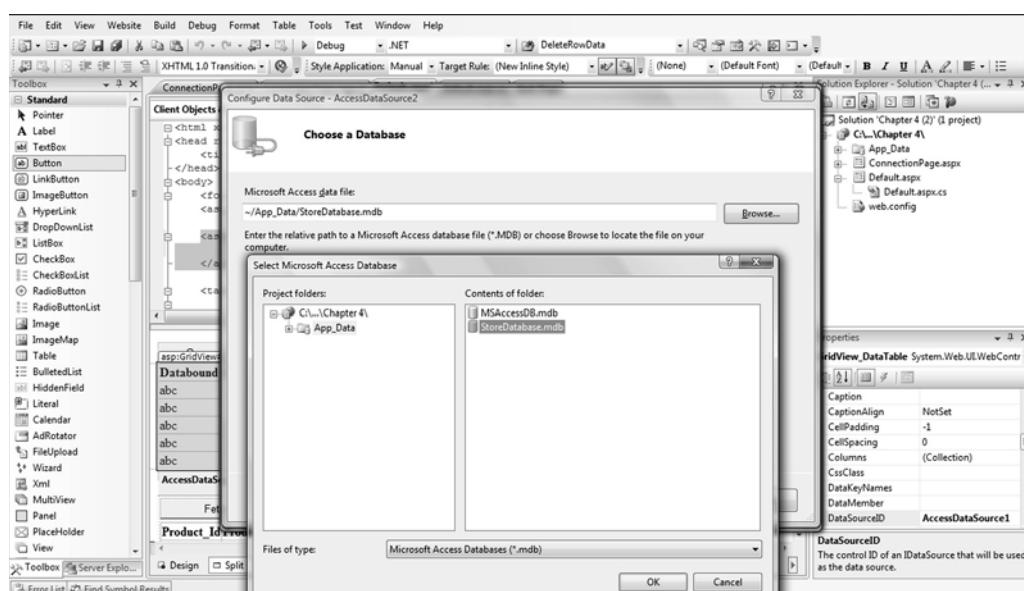
Choose a data source type screen



4. Browse to the StoreDatabase.mdb file, click OK as shown in Figure 4-7. Then, click the Next button.

Figure 4-7

The Choose a Database dialog box



5. A list of tables in the StoreDatabase.mdb is displayed in the Name drop-down list box, as shown in Figure 4-10. Select the table that you want to display in the GridView. Then, select the columns that you want to select in the GridView.
6. Click Next and then Finish.
7. The AccessDataSource1 database gets created, which connects GridView to the database table directly. The following code gets generated in the Default.aspx page:

```
<asp:AccessDataSource ID="AccessDataSource1" runat="server"
DataFile="~/App_Data/StoreDatabase.mdb"
SelectCommand="SELECT * FROM [Products]">>
</asp:AccessDataSource>
```

8. The data for the Products table is displayed in the GridView control.

CERTIFICATION READY?
Implement a DataSource Control.

3.4



ESTABLISH CONNECTIONS USING SQLCONNECTION

You can use SQL Server Express to establish a connection using `SqlConnection` in SQL Server 2005:

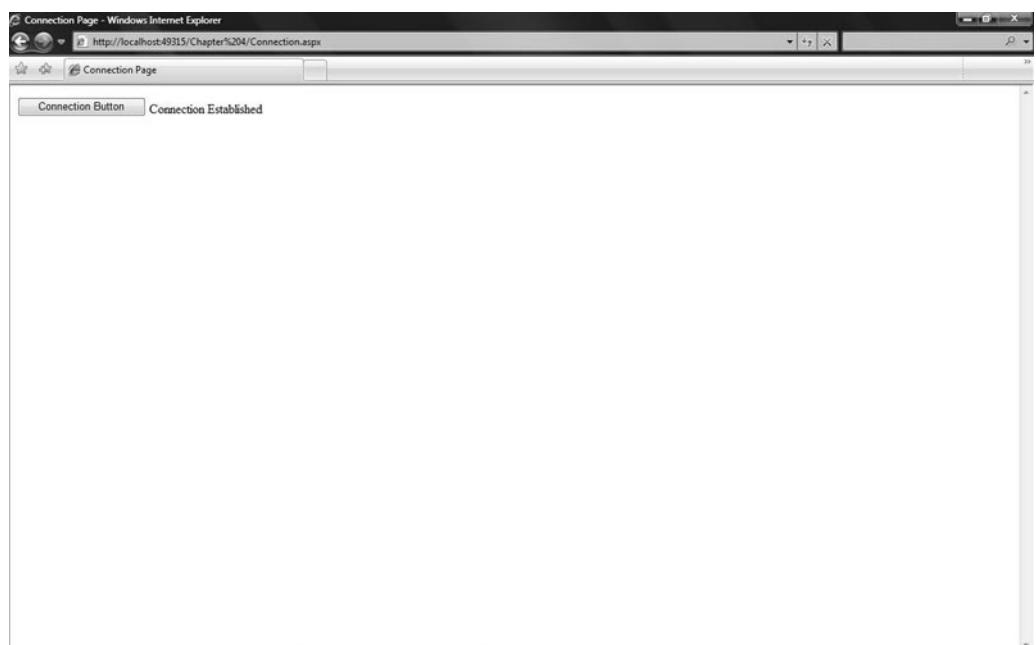
1. Open a new Web site from Visual Studio 2008, and name it MyConnection.
2. Add a button on the default.aspx page and label it Display.
3. Add two labels on the default.aspx page, one to display the open connection message and other to display the close connection message.
4. Add using `System.Data.SqlClient` at the top of the page.
5. Write the following code on the `Button1_Click` event:

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Make a connection string
    string connString = @"server =.\sqlExpress;integrated security
= true";
    // Create a connection
    SqlConnection conn = new SqlConnection(connString);
    try
    {
        // Open the connection
        conn.Open();
        Label1.Text = "Connection opened";
        // Close the connection
        conn.Close();
        Label1.Text = "Connection closed";
    }
    catch (SqlException exp)
    {
        // Display error or handle the error
        Label1.Text = "Error" + exp;
    }
}
```

6. Execute the code by pressing F5. The default screen will appear as shown in Figure 4-8.

Figure 4-8

Default screen



7. Click the Display button to see the labels showing the messages of open connection and close connection.



READ A CONNECTION STRING FROM THE WEB.CONFIG FILE

1. Follow the first four steps from the previous code example.
2. Add the following lines in your web.config file:

```
<connectionStrings>
<add name="SqlConn" connectionString="@server =
.\sqlexpress;integrated security = true;" />
</connectionStrings>
```

3. Add the following code in the Button1_Click event of the Display button:

```
protected void Button1_Click(object sender, EventArgs e)
{
    // Make a connection string
    string connString = ConfigurationManager.ConnectionStrings
    ["SQLConn"].ConnectionString;

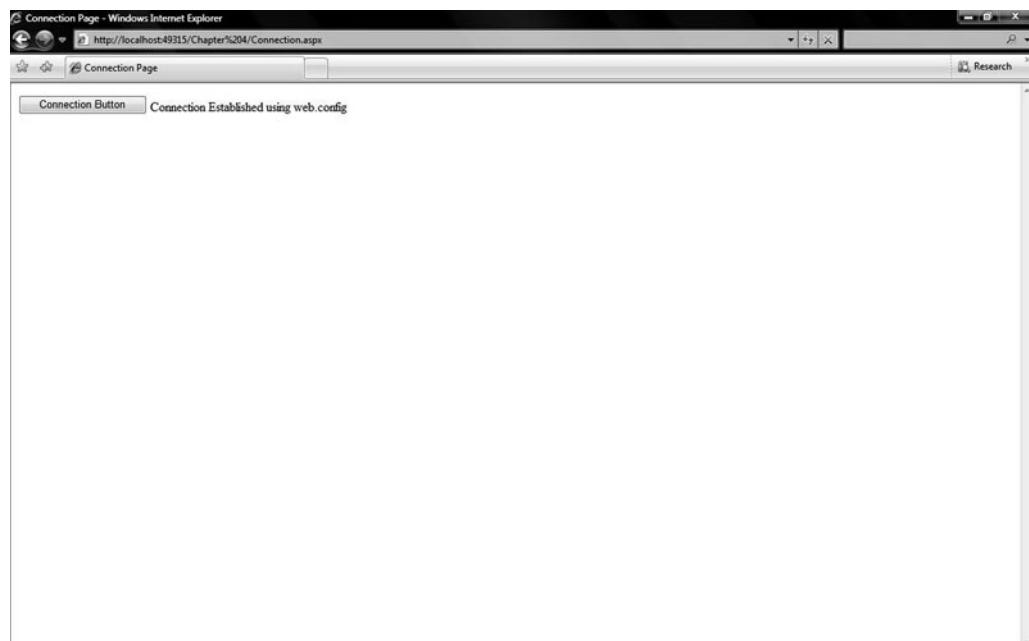
    // Create a connection
    SqlConnection conn = new SqlConnection(connString);

    try
    {
        // Open the connection
        conn.Open();
        Label1.Text = "Connection opened";
        // Close the connection
        conn.Close();
        Label1.Text = "Connection closed";
    }
    catch (SqlException exp)
    {
        // Display error or handle the error
        Label1.Text = "Error" + exp;
    }
}
```

4. Run the application by pressing F5 and clicking the Display button. The result page appears, as shown in Figure 4-9.

Figure 4-9

Result page



SKILL SUMMARY

Data access is an important component of any application. This lesson discussed ADO.NET and its two categories. This lesson also explained the data providers supported by ADO.NET in brief. The ADO.NET data providers connect to different types of databases, like SQL Server, Oracle, and Microsoft Access. These data providers are used to access data from these databases.

Data access can be connected and disconnected. You learned that the data providers are the key components of connected data access. The data provider for a database has its own set of connection commands and `DataReaders`. You saw how a connection string is written for different data providers with the help of sample code. In addition, you learned to use the ADO.NET command object in association with the connection object.

Disconnected data access can be considered to be the strength of ADO.NET. The key components of disconnected datasets are `DataAdapters`, `DataTables`, and `DataSet`. The `DataAdapter` is specific to a data provider and fills the `DataSet` with the help of the `Fill` command. The `DataSet` retains the data for the application to use without staying connected to the database. The `DataSet` is a collection of `DataTables` and maintains relationships between the tables.

You also learned about manipulating data using ADO.NET in this lesson. You learned how to update the database using the `UpdateCommand`, insert records into the database using the `InsertCommand`, and delete existing data from the database using the `DeleteCommand`.

For the certification examination:

- Understand the data providers supported by ADO.NET.
- Implement connected data access.
- Implement disconnected data access.
- Use `DataSet` and `DataReader` objects to manipulate data.
- Create and use database connections using the `ObjectDataSource` and `SqlDataSource` controls.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. A referential constraint that helps build relationship between two tables is known as a _____ key.
2. To open a connection on Microsoft SQL Server 7.0, you use the _____ object.
3. The `Close` method closes the connection temporarily, but the _____ method permanently destroys the connection.
4. The ADO.NET object model provides two command objects derived from the `DbCommand` class, `SqlCommand` and _____ to retrieve data from the data sources.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|-------|---|
| T F | 1. <code>SqlCommand</code> is a traditional way of command execution and usually returns a cursor that traverses through the rows in a table. |
| T F | 2. Whenever you store connection strings in the <code>web.config</code> file, you need to recompile the code every time you make changes. |
| T F | 3. <code>DataSet</code> allows access to the data only in the connected mode. |
| T F | 4. The <code>DataRow</code> object maintains three versions of data. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. Which of the following providers provide data access to data sources such as SyBase, Microsoft Access, DB2/400, and SQL Server 6.5 and earlier versions?
 - a. OLEDB
 - b. ODBC
 - c. SQL Server
 - d. Oracle

2. Which of the following managed data providers helps populate a `DataSet` object that is used to update a data source?
 - a. Transaction
 - b. DataAdapter
 - c. DataReader
 - d. Parameter

3. Which of the following managed providers automatically generates the command properties of a `DataAdapter`?
 - a. CommandBuilder
 - b. ConnectionStringBuilder
 - c. DataAdapter
 - d. Command

4. Which of the following string parameters of connection pooling causes the request for a new connection to be processed from the pool?
 - a. Connection Timeout
 - b. Connection Reset
 - c. Enlist
 - d. Pooling

Review Questions

1. What is the primary difference between the two command objects derived from the `DBCommand` class?

2. In which situations, will you use GUID?

■ Case Scenarios

Scenario 4-1: Using Connection Strings

You are creating an application for a drugstore. The store has multiple outlets at different cities and uses two databases at their stores for licensing issues—one with SQL Server 2005 and the other with MS Access. Your application should be compatible with both. How will you configure the connection string for the application per the requirement with minimal changes?

Scenario 4-2: Using Paging

For your drugstore application, there is a requirement to display transactions on a daily, monthly, and yearly basis. The UI displays the records in a grid of fixed size. The maximum number of records that can be displayed at a time is 20. The challenge lies in retrieving the annual data. How will you handle the load of huge data on your application?



Workplace Ready

Data Access Modes

You are developing an application to display a railway timetable on an application. While designing the application, you need to decide on the data connection type. You need to decide whether to design a connected data access mode or a disconnected data access mode. You should decide whether to use stored procedure or inline query, where to use DataReader, DataAdapter, and Dataset. You need to decide on these questions before you finalize the data layer of your application. How will you decide this?

Using XML and LINQ

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Using XML.	Read and write XML data.	3.1
Accessing Data with LINQ and SQL.	Implement a DataSource control.	3.4

KEY TERMS

Document Object Model (DOM)

Extensible Markup Language (XML)

Extensible Stylesheet Language (XSL)

language-integrated query (LINQ)

You have a database that stores your company's employee information. The database structure is quite large, and it holds a huge amount of data. You have been asked to utilize the information with multiple formats, such as placing it on to a newly developed web portal in a PDF document and in a Microsoft Word document. The database holds a large number of tables that contain a number of fields from which the data needs to be extracted. Without using XML, you will be required to reformat the same set of information in different ways to represent it in multiple outputs. With the help of XML, you can store the information and also present the output in different forms such as an HTML document or a PDF file.

■ Introducing XML



THE BOTTOM LINE

When developing a web application, it is recommended that you store data in .NET classes and objects instead of accessing the database for information every time. Storing data in .NET enables easy execution of logical conditions and passing of objects to other layers within the application. **Language-integrated query (LINQ)** and SQL provide you with a way to store database information in .NET by using object/relation mapping.

Extensible Markup Language (XML) is being widely used across different types of applications on a wide variety of platforms. It is derived from Standard Generalized Markup Language (SGML) and carries most of its characteristics. XML is a recommendation of the World Wide Web Consortium (W3C).

XML is entirely platform independent, which means you do not need any special applications to extract data stored in XML documents. One of the biggest advantages of XML is that it can be used with different applications without having to redefine the code structure in the XML documents. However, sometimes you might have to configure your application to become XML aware. When there is a need to represent data from a single source, XML becomes the ideal choice because of its adaptability to large applications.

XML is simple to understand and write. You create an XML document to structure, store, and transport information. The tags in an XML document are not predefined. You must define your own tags, which should be self-descriptive. Any complex information can also be stored in nested structures.

The **Extensible Stylesheet Language (XSL)** document is responsible for processing and formatting information stored in the XML document. Presentation of the information is controlled by an XSL style sheet. When you want to represent the information in a specific format, for example HTML, you will need the XML and XSL documents to generate the required output.

Using XML Namespaces and Classes

The `System.Xml` namespace provides the XML classes.

To use these classes, you need to add a `System.Xml.dll` reference and the `System.Xml` directive in your code. The `System.Data` assembly extends the `System.Xml` namespace, and the namespace includes the `XmlDataDocument` class. Therefore, if you are using this class in your application, you must also add a `System.Data.dll` reference.

ASP.NET offers various XML classes, each of which provides a variety of functionalities. It is very important that you determine which class to use in different situations.

USING XMLDOCUMENT AND XMLDATADOCUMENT CLASSES

There are two main ways to manipulate XML data: using stream-based XML processing or using in-memory XML processing. Stream-based XML processing is typically used with large XML files that do not fit in memory. For these, you need to use the two stream-based classes `XmlTextReader` and `XmlTextWriter`. In-memory XML processing is more convenient to use when you do need to manipulate large amounts of information in an XML document. In this case, you can use the `XmlNode` and `XmlDataDocument` classes.

The `XmlDataDocument` class is based on the `XmlNode` class, which is an implementation of the **Document Object Model (DOM)** Level 1 and DOM Level 2 that are defined by W3C. The main function of the `XmlNode` class is to use DOM to define an in-memory representation of XML documents. With the use of DOM, you get the flexibility to manipulate XML documents or to navigate through them.

Both the classes, `XmlNode` and `XmlDataDocument`, contain similar methods for executing common operations, such as navigating through XML nodes and editing them. The key methods used by the `XmlNode` and `XmlDataDocument` classes are:

- **GetElementByID:** Used for locating a specific element using its ID. When there is more than one match, only the first match is returned. If there are no matches, `NULL` is returned.
- **GetElementsByTagName:** Used for locating an object with its descendent elements by using the element's name. The output of this method is an `XmlNodeList` object.
- **Create<NodeType>:** Used for creating an `XmlNode` in an XML document. You can use three different types of `Create` methods for creating different types of nodes.
- **CloneNode:** Used for creating the clone of an XML node. You can either create a deep or shallow clone. If `deep` is set to `true`, the referenced node is cloned with all the child nodes. If `deep` is set to `false`, the referenced node is cloned without the child nodes.
- **InsertBefore:** Used for inserting an XML node before a specified node. If the XML node already exists in the tree, it is removed from its current location and inserted at the new specified location.
- **InsertAfter:** Used for inserting an XML node after a specified node. If the XML node already exists in the tree, it is removed from its current location and inserted at the new specified location

- **Load:** Used for loading an XML document from a specified source, which can be Stream, XMLReader, a URL, or a TextReader.
- **LoadXml:** Used for loading an XML document from a specified string.
- **ImportNode:** Used for importing a node from another `Xm1Document`. This method is typically used when you have a specific node in another XML document and you want to use it in your application. When you modify the node after importing, the source node will not get affected. You can copy a node with all its child nodes by setting the `deep` value to `true`.
- **PrependChild:** Used for inserting a node at the beginning of a child node list. If the XML node already exists in the tree, it is removed from its current location and inserted at the new specified location.
- **ReadNode:** Used for creating a node from the information in an `XMLReader` object.
- **RemoveAll:** Used for removing child nodes of the specified parent node.
- **RemoveChild:** Used for removing a specific child node of a specified node. The removed child node name is returned if it is removed successfully. If not, the value `NULL` is returned.
- **ReplaceChild:** Used for replacing a child node with another node. If the new child node is already present in the document, the existing new child node is first removed, and then, the new child node is added to the specified location. The name of the replaced child node is returned if the replace operation is executed successfully. If not, the value `NULL` is returned.

TAKE NOTE *

There are other methods for the `Xm1Document` and `XMLDataDocument` classes mentioned previously. For information on the methods of these two classes, refer to the section on `System.XML` namespace on MSDN.

USING THE XPATHNAVIGATOR CLASS

The `XPathNavigator` class is part of the `System.Xm1.XPath` namespace and is used to load complete information of an XML document into memory in one shot. It uses a cursor-based approach that allows you to navigate to specific nodes.

USING THE XPATHDOCUMENT CLASS

The `XPathDocument` class is mainly used to execute XPath queries and perform XSL transformations. It uses the XPath data model to build a read-only, in-memory tree structure. You can use the `XPathNavigator` interface to navigate through the tree structure.

USING THE XMLCONVERT CLASS

The `Xm1Convert` class is used to convert from Common Language Runtime (CLR) datatypes to XML Schema Definition (XSD) datatypes and vice versa. You can also use this class to check the validity of the XML names that are derived from the source XML document.

USING THE XSLTRANSFORM CLASS

The `Xs1Transform` class uses an XSL style sheet to transform XML documents using the `Load` and `Transform` methods. These are steps to perform an XML document transformation:

1. Create an object of the `Xs1Transform` class.
2. Load the style sheet using the `Load` method.
3. Execute data transformation using the `Transform` method.

USING THE XMLNODEREADER CLASS

The `XmlNodeReader` class provides an interface to read data from an XML document. This class provides a read-only, forward-only access for retrieving information from the nodes of an XML document.

USING THE XMLREADER CLASS

The `Xm1Reader` class provides a forward-only access for retrieving information from the nodes in an XML document. The XML data can be retrieved either from a data file or from a stream. You can read XML data from a specific node by using the methods and properties of the `Xm1Reader` class.

USING THE XMLTEXTREADER CLASS

The `XmlTextReader` class is based on the `XMLReader` class and is another method used to read XML data either from data files or streams. `XmlTextReader` follows a noncached method of reading data and reads only those files that adhere to the W3C XML 1.0 standards.

USING THE XMLTEXTWRITER CLASS

The `XmlTextWriter` class provides a noncached method of writing XML data on to files or streams. It writes by using the forward-only method to only those files that adhere to the W3C XML 1.0 standards.

Working with XML Documents

ASP.NET provides various ways of working with XML data. You can create a new XML file, write data to it, read data from it, and transform the XML data.

In this section, you'll focus on some of the methods used for creating a new XML file, reading and writing to XML files, searching for data from an XML file, and transforming XML data.

CREATING AN XML DOCUMENT

The `X XmlDocument` class is used to create a new XML document and add data to it. The following two methods are used to add nodes to an `X XmlDocument` object:

- **CreateElement:** Creates a new element in an XML document using the `XmlElement` class.
- **CreateAttribute:** Creates an attribute and adds it to the `XmlDocument`.

The `X XmlAttributeCollection` class defines a single `Attributes` property of an `XmlElement` class and a collection of attributes for an `XmlElement` node. The following code example shows you how to create an XML document and save it to a file:

```
protected void Page_Load(object sender, EventArgs e)
{
    XmlDocument xDoc = new XmlDocument();

    // Create the version tag for XML
    xDoc.AppendChild(xDoc.CreateXmlDeclaration
("1.0", "utf-8", null));

    // Create the Root Node and append it to XmlDocument
    XmlElement el = xDoc.CreateElement("Root");
    xDoc.AppendChild(el);
    for (int i = 1; i < 4; i++)
    {
        XmlElement childElement = xDoc.CreateElement("Child");
        XmlAttribute childAttrib = xDoc.CreateAttribute("ID");
        childAttrib.Value = i.ToString();

        childElement.Attributes.Append(childAttrib);
        // Append element into root element
        el.AppendChild(childElement);
    }
    // Save XmlDocument
    xDoc.Save("C:\\Test XmlDocument.xml");
    Response.Write("File Test XmlDocument.xml is saved.");
}
```

The newly created file `Test XmlDocument.xml` will contain the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<Root>
    <Child ID="1" />
```

TAKE NOTE*

When you save the XML document to the disk file, you must have write access to the specified directory. Otherwise, the application will throw an exception.

```
<Child ID="2" />
<Child ID="3" />
/>/Root>
```

PARSING AN XMLDOCUMENT CLASS USING THE DOM

To be able to retrieve and modify data stored in an XML document, you need to parse the XML. You can parse an XML document class by recursively iterating through all the nodes. The following code example shows you how to parse an XML document class when the Parse button on a web page is clicked:

```
protected void Parse_Click(object sender, EventArgs e)
{
    XmlDocument xDoc = new XmlDocument();
    xDoc.Load("C:\\\\TestXmlDocument.xml");
    IterateNodes(xDoc.DocumentElement);
}

private void IterateNodes(XmlNode xmlNode)
{
    // Start iterating with level 0 node IterateNodes(xmlNode, 0);
}

private void IterateNodes(XmlNode xmlNode, int level)
{
    string s = String.Format("{0}<b>Type:</b>{1} <b>Name:</b>{2}<br>Attr:</b>", new string('-', level), xmlNode.NodeType,
    xmlNode.Name);
    foreach (XmlAttribute attr in xmlNode.Attributes)
    {
        s += String.Format("{0}={1} ", attr.Name, attr.Value);
    }
    Label1.Text += s + "<br/>";
    foreach (XmlNode n in xmlNode.ChildNodes)
    {
        IterateNodes(n, level + 1);
    }
}
```

In this code, when the Parse button is clicked, an XML file is loaded, and the overloaded `IterateNodes` method is called. The first call passes the root node of the `xDoc` object. The recursive call passes the recursion level. Every time the `IterateNodes` method is executed, the information of each node is displayed as shown here:

```
Type:Element Name:Root Attr:
-Type:Element Name:Child Attr:ID=1
-Type:Element Name:Child Attr:ID=2
-Type:Element Name:Child Attr:ID=3
```

You can also use the `XPathNavigator` class to recursively parse an XML document. This class uses the XPath data model and provides methods to use XPath queries for navigation. The following code example shows you how to use the `XPathNavigator` class:

```
protected void Button1_Click(object sender, EventArgs e)
{
    XmlDocument xDoc = new XmlDocument();
    xDoc.Load("C:\\\\TestXmlDocument.xml");
    XPathNavigator xpathNav = xDoc.CreateNavigator();
    IterateNodes(xpathNav);
}

private void IterateNodes(XPathNavigator xpathNav)
```

```
{  
    // Start recursive iteration with level 0  
    IterateNodes(xpathNav, 0);  
}  
private void IterateNodes(XPathNavigator xpathNav, int level)  
{  
    string s = String.Format("{0} <b>Type:</b>{1}  
<b>Name:</b>{2} <b>Attr:</b>", new string('-', level), xpathNav.  
NodeType, xpathNav.Name);  
    if (xpathNav.HasAttributes)  
    {  
        xpathNav.MoveToFirstAttribute();  
        do  
        {  
            s += string.Format("{0}={1}", xpathNav.Name,  
xpathNav.Value);  
        } while  
(xpathNav.MoveToNextAttribute());  
        xpathNav.MoveToParent();  
    }  
    // You need to put a Label control named Label1 on the  
    web page  
    Label1.Text += s + "<br>";  
    if (xpathNav.HasChildren)  
    {  
        xpathNav.MoveToFirstChild();  
        do  
        {  
            IterateNodes(xpathNav, level + 1);  
        }  
        while (xpathNav.MoveToNext());  
    }  
}
```

In this code example, the same recursive approach has been used; however, different methods of the `XPathNavigator` class have been employed.

In the previous code:

- The `HasAttributes` property has been used to access the attributes of the node.
- If the current node has any attributes, the property is set to `true`.
- The `MoveToFirstAttribute` and `MoveToNextAttribute` methods have been used to navigate through the attributes.
- After the attribute list of a specific node is complete, the `MoveToParent` method has been used to move back to the parent node.
- If the `HasChildren` property returns `true`, it indicates that the current node has child nodes.
- To navigate through the child nodes, the `MoveToFirstChild` and `MoveToNext` methods have been used.
- After navigation through all the child nodes is complete, the `MoveToParent` method has been used to move back to the parent.

The output of the code example is shown here:

```
Type:Root Name: Attr:  
- Type:Element Name:Root Attr:  
- Type:Element Name:Child Attr:ID=1  
- Type:Element Name:Child Attr:ID=2  
- Type:Element Name:Child Attr:ID=3
```

SEARCHING AN XMLDOCUMENT OBJECT USING THE DOM

The `XmlDocument` class provides the `GetElementByID` and `GetElementsByTagName` methods to search for data from an XML document. The `GetElementByID` method searches for an `XmLElement` based on its ID as specified in a DTD document. In case there are multiple elements with the matching ID, the first matching element will be returned:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE root[
    <!ELEMENT root ANY>
    <!ELEMENT child ANY>
    <!ATTLIST child ChildID ID #REQUIRED]>
<root>
    <child ChildID="ref-1"></child>
    <child ChildID="ref-2"></child>
    <child ChildID="ref-3"></child>
</root>
```

In this code example, `ChildID` is defined as an ID datatype that must begin with a character, underscore, or colon.

The following code example shows you how to use the `GetElementByID` method that returns the node with an ID “ref-2”:

```
XmLDocument xDoc = new XmLDocument();
xDoc.Load("C:\\\\Sample.xml");
XmlNode node = xDoc.GetElementByID("ref-2");
```

The `GetElementByTagName` method searches the entire document based on the name of an element. This method returns an `XmLNodeList` that contains a list of all matching elements. The following line of code returns all the elements with the tag name `child`:

```
XmLNodeList list = xDoc.GetElementsByTagName("child");
```

Another method of the `XmLDocument` class that you can use to search an XML document is `SelectSingleNode`. This method takes an XPath query as an argument and returns the first element that matches the XPath expression. The `SelectSingleNode` method does not need a DTD and can perform an XPath search on any element or attribute, as shown in the following line of code:

```
XmlNode node = xDoc.SelectSingleNode("//child[@ChildID='ref-2']");
```

You can also use the `SelectNodes` method to perform an XPath lookup for elements in an XML document. This provides more querying flexibility. A search using the `SelectNodes` method returns an `XmLNodeList` that contains the list of all elements that match the XPath query. You need to pass an XPath query to this method.

TAKE NOTE*

The `SelectNodes` method provides you more querying flexibility when compared to the `GetElementsByTagName` method because the `GetElementsByTagName` method is limited to a tag name.

SEARCHING FOR XPATH DOCUMENTS USING THE XPATHNAVIGATOR CLASS

The `XPathNavigator` class works like the `XmLDocument` class and offers more search flexibility when compared to the DOM. The methods in `XPathNavigator` class use XPath queries to search for elements. You can create a read-only `XPathNavigator` object from an `XmLDocument` object using the `CreateNavigator` method. You can also create editable `XPathNavigator` objects from `XPathDocument` objects using fewer resources. The following code example shows the use of `XPathNavigator` to search for XPath documents:

```
XmLDocument xDoc = new XmLDocument();
xDoc.Load(@"C:\\\\Sample.xml");
```

```

XPathNavigator xpathNav = xDoc.CreateNavigator();
String expr ="//child[@ChildID='ref-2']";
string s;
XPathNavigator nav = xDoc.CreateNavigator();
XPathNodeIterator iterator = nav.Select(expr);
XPathNavigator navResult = iterator.Current;
while(iterator.MoveNext())
{
    s = String.Format("<b>Type:</b>{0} <b>Name:</b>{1} ", 
navResult.NodeType, navResult.Value);
    if (navResult.HasAttributes)
    {
        navResult.MoveToFirstAttribute();
        s += "<b>Attr:</b>";
        do
        {
            s += String.Format("{0}={1}", navResult.Name,
navResult.Value);
        }
        while (navResult.MoveToNextAttribute());
    }
    Label1.Text = s + "<br>";
}

```

TAKE NOTE *

The **XPathNavigator** class is preferred over other classes when you want to retrieve a sorted list of nodes. To sort the list of objects, you need to compile the XPath query to an **XPathExpression** object, and then, sort the compiled expressions.

This code example returns an element with the ChildID value “ref-2.” The **Select** method is called with the query string, which returns the **XPathNodeIterator** object. This object contains the properties of the current node and is an **XPathNavigator** datatype.

WRITING A FILE USING THE XMLTEXTWRITER CLASS

You can use the **XmlTextWriter** class to write an XML document to a stream without storing the XML document in memory all at once. The following code example shows you how to do this:

```

XmlTextWriter writer = new XmlTextWriter("C:\\\\Employee.xml",
System.Text.Encoding.UTF8);
writer.Formatting = Formatting.Indented;
writer.Indentation = 5;
writer.WriteStartDocument();
writer.WriteComment("XmlTextWriter");
writer.WriteStartElement("Employees");
writer.WriteStartElement("Employee");
writer.WriteString("EmpID", "1");
writer.WriteString("Name", "Joe");
writer.WriteString("Salary", XmlConvert.ToString(40000));
writer.WriteEndElement();
writer.WriteStartElement("Employee");
writer.WriteString("EmpID", "2");
writer.WriteString("Name", "Robin");
writer.WriteString("Salary", XmlConvert.ToString(43000));
writer.WriteEndElement();
// End of Employees
writer.WriteEndElement();
writer.Close();

```

This code performs the following tasks:

- Creates an **XmlTextWriter** object, which accepts the file name and the encoding for the XML file as parameters
- Formats the XML document by setting the **Formatting** and the **Indentation** properties

3. Writes the XML declaration to the file by using the `WriteStartDocument` method
4. Writes a comment by using the `WriteComment` method
5. Writes the element by using the `WriteStartElement` method
6. Completes the element by calling the `WriteEndElement` method
7. Saves the file using the `Close` method

The output is an XML file as shown:

```
<?xml version="1.0" encoding="utf-8"?>
<!-XmlTextWriter->
<Employees>
  <Employee EmpID="1" Name="Joe" Salary="40000" />
  <Employee EmpID="2" Name="Robin" Salary="43000"/>
</Employees>
```

READING AN XML FILE USING THE XMLTEXTREADER CLASS

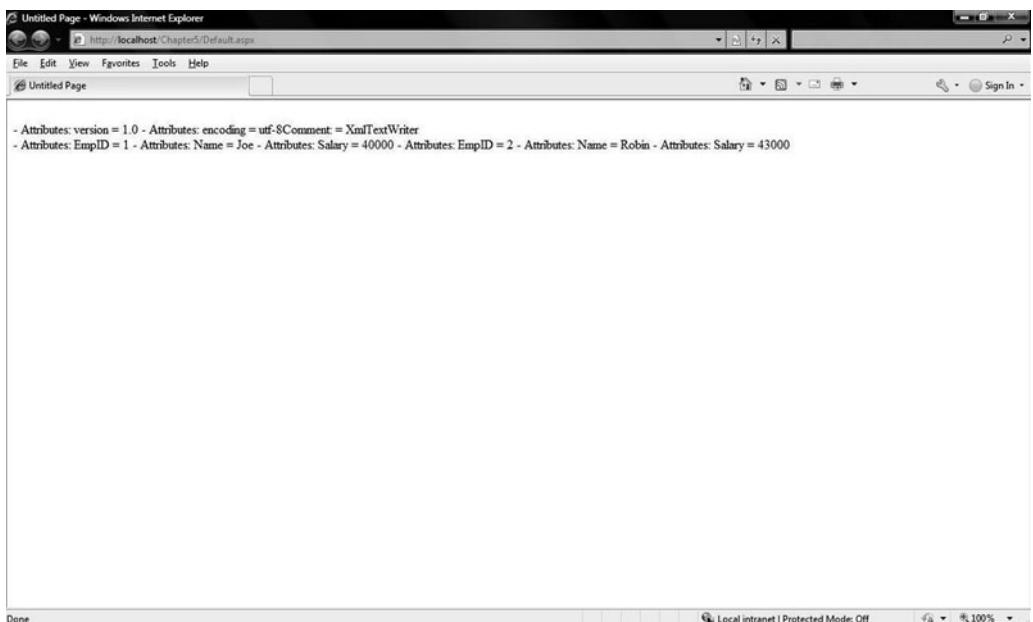
The `Xm1TextReader` class reads one node at a time in an XML file. The access to the XML document is forward-only, which means that the document will be read from the specified starting point onwards. Although, using an `Xm1TextReader` object is simple, it does not give you flexibility of randomly accessing data. The following code example reads an XML file:

```
Xm1TextReader reader = new
Xm1TextReader("C:\\Employee.xml");
while (reader.Read())
{
    switch (reader.NodeType)
    {
        case Xm1NodeType.Comment:
            {
                string s = String.Format("{0}: {1} =
{2}<br>", reader.NodeType, reader.Name, reader.Value);
                Label1.Text += s;
                break;
            }
        case Xm1NodeType.Text:
            {
                string s = String.Format(" - Value:
{0}<br>", reader.Value);
                Label1.Text += s;
                break;
            }
    }
    if (reader.HasAttributes)
    {
        while (reader.MoveToNextAttribute())
        {
            string s = String.Format(" - Attributes: {0}
= {1}", reader.Name, reader.Value);
            Label1.Text += s;
        }
    }
}
reader.Close();
```

This code opens the `Employee.xml` file and reads all nodes, one node at a time. This code also performs a check on each node for its type. Depending on the type of node, relevant information is displayed, as shown in Figure 5-1.

Figure 5-1

Displaying node information



MODIFYING AN XML DOCUMENT

As discussed earlier, XML documents hold data in the form of nodes. After creating an XML document, if you need to update the content within the document, you can use the `X XmlDocument` class. This class uses the XML DOM parser. This model can also be used for adding, deleting, or updating the data nodes present in the XML document. To insert data into an XML document, you need to perform the following steps:

1. Create the node.
2. Define the location of the node.
3. Insert the node.

The following code example demonstrates how to insert a new node in the Sample.xml file:

```

XmlElement e1 = xDoc.CreateElement("new");
XmlNode node = xDoc.SelectSingleNode("//child[@ChildID='ref-1']");
node.ParentNode.InsertAfter(e1, node);
  
```

Deleting a node follows the same process as inserting a new node. However, instead of following the last step to insert a node, you will delete it. The following lines of code show you how to locate and delete a node from the Sample.xml file:

```

XmlNode node = xDoc.SelectSingleNode("//child[@ChildID='ref-2']");
node.ParentNode.RemoveChild(node);
  
```

VALIDATING AN XML DOCUMENT

After creating an XML document, you must validate it against the DTDs or XML schema, which contains the standard structure of XML elements and their attributes. You can use the `X XmlDocument` class to first read through the XML document. Then, you can perform the validation by creating an `XMLReaderSettings` object. After creating the object, you also need to set its properties appropriately.

The following code example shows you how to validate an XML document:

```

protected void Button5_Click(object sender, EventArgs e)
{
    if (ValidateDocument(@"C:\Sample.xml"))
    {
        Label1.Text = "This is a Valid XML";
    }
}
  
```

```

        }
    else
    {
        Label1.Text = "This is invalid XML.";
    }
}
private bool ValidateDocument(string fileName)
{
    XmlReaderSettings setting = new XmlReaderSettings();
    setting.ValidationType = ValidationType.DTD;
    setting.ProhibitDtd = false;
    XmlReader reader = XmlReader.Create(fileName, setting);
    XmlDocument xdoc = new XmlDocument();
    try
    {
        xdoc.Load(reader);
        return true;
    }
    catch (Exception)
    {
        return false;
    }
    finally
    {
        reader.Close();
    }
}

```

CERTIFICATION READY?

Read and write XML data.

3.1

■ Accessing Data with LINQ and SQL



THE BOTTOM LINE

At times, you need to manipulate the data stored in a database. Typically, modern programming languages use objects, and databases use rows, to store data. The representation of the data from rows into objects is a tedious task. As programmers, you want to manipulate the data stored in the database by using objects and still ensure that the data in the rows is intact. To bridge this gap, Microsoft introduced LINQ to SQL, which is a .NET technology. It is a part of the Microsoft .NET Framework 3.5, and its primary function is to bridge the gap between how databases store data and how programming languages, like C# and Visual Basic, need to interpret that data with their native syntaxes while still maintaining data consistency.

The LINQ to SQL technology is part of ADO.NET and is built to work with an SQL Server database to enable LINQ-style programming. LINQ to SQL translates the programming syntaxes into the SQL queries for data manipulation and retrieves data from rows in the form of the objects. It uses object relational (O/R) mapping that helps it connect to the database elements.

Creating LINQ and SQL Entities

LINQ defines the standard query operators for traversal, filter, and projection operations to be expressed in a direct yet declarative way in the .NET supported programming language. The standard query operations allow queries to be applied to any type of information source based on `IEnumerable <T>` interface. LINQ also allows you to use third-party domain-specific operators that are more appropriate for a specific domain.

The query architecture of LINQ provides implementations that support both SQL and XML data. The LINQ to XML technology uses an easy-to-use, efficient, in-memory XML facility to provide XPath/XQuery functionality. The LINQ to SQL technology is built on SQL-based schema definitions into CLR type systems. This integration provides strong typing over the relational database while retaining the expressive power of the relational model.



WRITE LINQ CODE

To write LINQ code against the database:

1. Add a reference to the `System.Data.Linq` namespace, which contains the `DataContext` object. This step is important because the `DataContext` object is responsible for connectivity between the database and the O/R map.
2. Create an O/R map to connect objects to the data tables and columns. These objects are also known as entities or entity classes.
3. Use the `DataContext` object to establish connectivity between the database and the O/R map.
4. Connect to the database to write queries that manipulate data stored within the database once the connectivity is established.

Because LINQ to SQL is a part of ADO.NET, you have the flexibility of using objects that are written by using ADO.NET.

Mapping Objects to Relational Data

You can map objects to the relational data in different ways. Let's understand how.

You can map the objects using the Visual Studio Designer tool, command-line tool, or code editor.

MAPPING WITH THE VISUAL STUDIO DESIGNER TOOL

The Visual Studio Designer tool helps you to build an O/R map without writing a single line of code. You can simply drag and drop the database entities into the designer surface. In Visual Studio, you first need to open the `.dbml` file. Then, you need to invoke Server Explorer to choose the database entities that you require. Now, you can drop the entities onto the designer surface to finish generating your O/R map.

The Designer tool detects the database entities and automatically generates the relevant code. The output is in XML form where objects representing tables and fields are generated. Apart from tables and fields, you can also represent database stored procedures in an O/R map by generating code that reads them as .NET methods. The end result will be a `.dbml` file and a code file with the extension `.vb` or `.cs`, depending on whether you choose Visual Basic or C#.



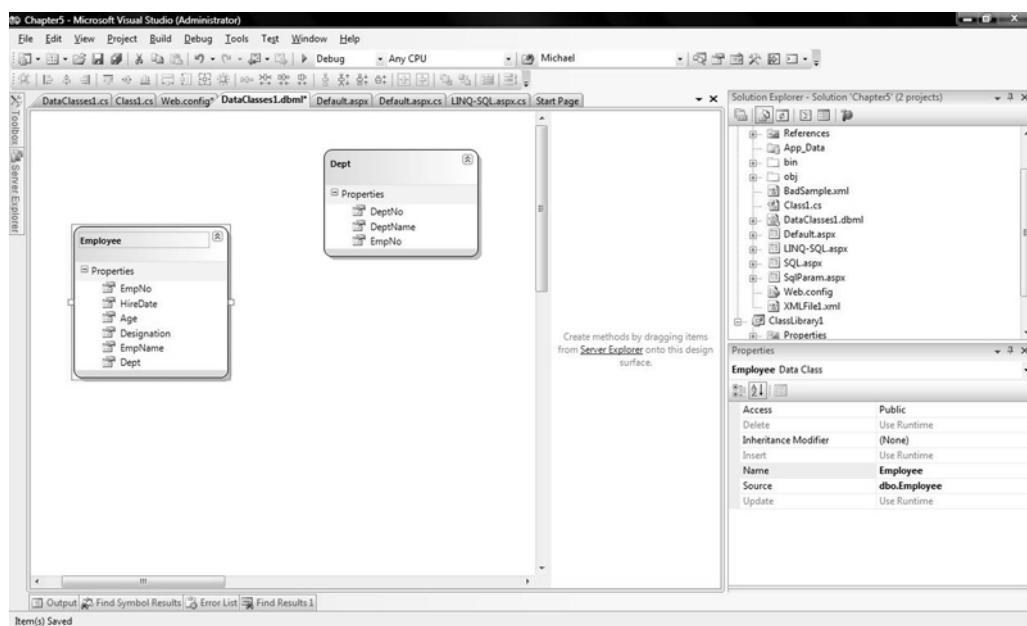
GENERATE AN O/R MAP

Suppose you are working with the master database and want to generate the O/R map for the Employee and Dept tables. To do this:

1. Right click Solution Explorer.
2. Select Add New Item.
3. From the newly opened Add New window, select LINQ to SQL classes, and click the Add button. This adds the `DataClasses1.dbml` file in your project.
4. From the Server Explorer, drag and drop the Employee and Dept tables as shown in Figure 5-2.

Figure 5-2

Server Explorer with data tables



This also generates an XML file for the .dbml file in the layout as shown next:

```
<?xml version="1.0" encoding="utf-8"?>
<ordesignerObjectsDiagram ds1Version="1.0.0.0"
absoluteBounds="0, 0, 11, 8.5" name="DataClasses1">
<DataContextMoniker Name="/DataClasses1DataContext" />
<nestedChildShapes>
<classShape Id="096170b3-80bb-40c6-8dcc-3e6be91f1959"
absoluteBounds="0.5, 1.125, 2, 1.9631982421875">
<DataClassMoniker Name="/DataClasses1DataContext/Employee" />
<nestedChildShapes>
<elementListCompartment Id="2988e8f7-95e3-4eea-8ddd-640041f234d0"
absoluteBounds="0.51500000000000012, 1.585,
1.9700000000000002, 1.4031982421875" name="DataPropertiesCompartment"
titleTextColor="Black" itemTextColor="Black" />
</nestedChildShapes>
</classShape>
<classShape Id="70205e39-2019-4103-8a04-f3ed5b5874bd"
absoluteBounds="2.875, 0.625, 2, 1.3862939453125">
<DataClassMoniker Name="/DataClasses1DataContext/Dept" />
<nestedChildShapes>
<elementListCompartment Id="6a1a9c30-1576-40d8-99ea-
827208c15ca6" absoluteBounds="2.89, 1.085, 1.9700000000000002,
0.8262939453125" name="DataPropertiesCompartment"
titleTextColor="Black" itemTextColor="Black" />
</nestedChildShapes>
</classShape>
</nestedChildShapes>
</ordesignerObjectsDiagram>
```

The generated code in the code-behind file for the dbml will be either in C# or Visual Basic based on the type of project. The code defines an object for each table. It also defines the properties within each object for each field in the table.

MAPPING WITH THE COMMAND-LINE TOOL

Apart from using the Visual Studio Designer tool, another option to create an O/R map is using the command-line tool SqlMetal.exe. Both tools provide the same output but can be used in different situations. The Designer tool is preferred over the command-line tool in

situations where you need a visual representation of databases. In cases where the databases are larger, the command-line tool SqlMetal.exe is found to be more helpful. To create an O/R map with SqlMetal.exe, you need to perform two steps:

1. Generate the .dbml file.
2. Generate the code-behind file.

The syntax for the invoking SqlMetal.exe is:

```
SqlMetal [options] [<input file>]
```

Some other options you can use with SqlMetal.exe to connect to the database include:

- /server:<name>
- /database:<name>
- /user:<name>
- /timeout:<seconds>
- /functions
- /language:<language>

MORE INFORMATION

There are other options that you can use with SqlMetal.exe. For information on the options and their uses, refer to the section on SqlMetal.exe (Code Generation Tool) on MSDN.

MAPPING OBJECTS TO ENTITIES IN THE CODE EDITOR

You can connect classes to database objects using code. This gives granular control but it is a cumbersome task.



MAP OBJECTS TO ENTITIES

To map objects to entities:

1. Create a class file, and at the top, add a reference to System.Data.Linq and System.Data.Linq.Mapping namespaces:

```
using System.Data.Linq;
using System.Data.Linq.Mapping;
```
2. Link the class to the table in the database using the Table attribute:

```
[Table(Name="Employee")]
public class Employee
```
3. Create the properties on the object that map to the database table columns using the Column attribute class. This class allows you to set the names of the database fields and variables, and whether the item is the primary key, as shown in the following code snippet:

```
[Column(IsPrimaryKey=true,Name="EmpID")]
public int EmpID
{ get; set; }
[Column(Name="EmpName")]
public string Name
{ get; set; }
```

Once you've defined the properties, you can use the custom O/R map class the way you use the generated class from the code generator.

Querying Data with LINQ to SQL

To query a database, you need to first create a connection to the database. This can be done using a `DataContext` object.

A `DataContext` object bridges the gap between database entities and the programming objects. You must create `DataContext` objects and establish a connection with the database only.

when you need to wrap the database for data manipulation. In addition, once manipulation is completed, you must immediately close the connection and release the `DataContext` object. It is never advisable to create a persistent `DataContext` object. To establish a connection after creating an instance of the `DataContext` class, you need to pass the database connection string to it. The following code shows you how to do this:

```
public class MyDataContext: DataContext
{
    public Table<Employee> Employees;
    public MyDataContext(string connection): base(connection)
    {
    }
}
```

The following code example shows you how to instantiate the `MyDataContext` class and execute a query against the table class:

```
string connectionString="Data Source=localhost\\SQLEXPRESS;Integrated Security=True";
MyDataContext context = new MyDataContext(connectionString);
var query = from employee in context.Employees
select employee;
GridView1.DataSource = query.ToArray();
GridView1.DataBind();
```

Performing Insert, Update, and Delete Using LINQ to SQL

After creating a connection between an O/R map and its database, you can start performing the standard data manipulation operations such as insert, modify, and delete.

By default, the changes made to a database are not written back to the database. You need to call the `SubmitChanges` of the `DataContext` object method to commit the changes. For example, if you insert a new employee record into the Employee table using the `InsertOnSubmit` method, this change should be submitted back to the database by using the `SubmitChanges` method. The following code example illustrates this:

```
string connectionString="Data Source=localhost\\SQLEXPRESS;Integrated Security=True";
MyDataContext context = new MyDataContext(connectionString);
Employee e1 = new Employee();
e1.Age = 39;
e1.EmpName = "Allen";
e1.HireDate = System.DateTime.Today;
context.Employees.InsertOnSubmit(e1);
context.SubmitChanges();
```

Similarly, you can perform a delete operation by using the `DeleteOnSubmit` method and submit changes using the `SubmitChanges` method. However, to perform an update operation, you need to use the `GetTable` method. This method helps in fetching the data to be modified from the database.

The following example illustrates how to update data:

```
string connectionString="Data Source=localhost\\SQLEXPRESS;Integrated Security=True";
MyDataContext context = new MyDataContext(connectionString);
var query = from employee in context.Employees
select employee;
Employee update = query.First();
update.Age = 43;
context.SubmitChanges();
```



CREATE AN XMLDOCUMENT

To create an XMLDocument:

1. Add a new web form.
2. In the code-behind file, put the following code:

```
 XmlDocument xDoc = new XmlDocument();
xDoc.AppendChild(xDoc.CreateXmlDeclaration("1.0", "utf-8", null));
XmlElement root = xDoc.CreateElement("Microsoft");
xDoc.AppendChild(root);

XmlElement product = xDoc.CreateElement("Product");
XmlAttribute name = xDoc.CreateAttribute("Name");
name.Value = "Microsoft Visual Studio";
XmlAttribute version = xDoc.CreateAttribute("Version");
version.Value = "9.0";
product.Attributes.Append(name);
product.Attributes.Append(version);

root.AppendChild(product);
xDoc.Save("C:\\\\Product.xml");
```

This will create a file named Product.xml in the C:\\\\ drive.

SKILL SUMMARY

In this lesson, you learned the advantages of XML and the various classes and namespaces in the .NET Framework that support XML. You also learned how to create a new XML document and how to parse existing documents using the `Xm1Document` class. In addition, you learned how to search for specific information in an XML document using different classes. You also learned about the `Xm1TextReader` and `Xm1TextWriter` classes that you use to read from and write data into an XML file. Then, you learned how to modify an XML document by adding new nodes. You also learned how to validate an XML document using DTD or schemas.

In addition, you learned about mapping objects to a relational database using LINQ to SQL. You also learned about the different ways of creating the O/R map and the method to retrieve data from a database using LINQ to SQL. You also learned how to insert, update, and delete data using LINQ to SQL.

For the certification examination:

- Use and validate XML documents and read and write data to an XML file.
- Create LINQ and SQL entities and execute standard database commands with LINQ and SQL.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ is the base class for the `Xm1DataDocument` class. It is used to represent relational data and can expose the data as an object of `DataSet`.
2. The `CreateElement` and _____ methods are used to add nodes to the `Xm1Document` object.
3. The _____ method searches an element in an XML document based on its ID.

4. You can use the command-line tool _____ to generate the .dbml file and O/R code for the database.
5. The _____ class reads an XML file node by node.

Matching

Match the term in Column 1 to its description in Column 2.

- | | |
|-------------------------|--|
| a. XMLReader | 1. Provides forward-only access to data in the <code>XmlDocument</code> or <code>XmlDataDocument</code> objects. |
| b. XMLTextReader | 2. Transforms an XML document using XSL style sheets. |
| c. XMLTransform | 3. Provides many static methods for conversion of XSD datatypes to CLR datatypes and vice versa. |
| d. XMLConvert | 4. Provides an object for reading and validating against the DTD, XDR, or XSD. |
| e. XmlNodeReader | 5. Provides noncached, forward-only access to the data in an XML document. |

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---------------------|---|
| T F | 1. XML is not a W3C recommendation. |
| T F | 2. XSLT is used to convert XML documents from one format to another. |
| T F | 3. LINQ is applicable to classes of type <code>IEnumerable<T></code> only. |
| T F | 4. The <code>SqlMetal</code> command doesn't work with the MS-Access database. |
| T F | 5. The <code>DataContext</code> class acts as the mediator between the relational element and the .NET class. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. XML stands for:
 - a. Extended Modeling Language
 - b. Extensible Modeling Language
 - c. Extensible Markup Language
 - d. Extended Markup Language
2. Which of the following methods is used to write a node and all of its descendants to another XML document?
 - a. `WriteTo`
 - b. `WriteContentsTo`
 - c. `Save`
 - d. `SaveAs`
3. Which class is used for converting XSD datatypes to CLR types?
 - a. `XmlTextReader`
 - b. `XmlDocument`
 - c. `XslTransform`
 - d. `XmlConvert`

4. What does O/R stand for?
 - a. Objects records mapping
 - b. Object relational mapping
 - c. Object recordset mapping
 - d. Object reader mapping
5. Why do you need to call the `SubmitChanges` method?
 - a. It is mandatory for creating the database connection.
 - b. It saves the changes back to the database.
 - c. It submits the changes by doing a postback to the web server.
 - d. It acts as a mediator between the objects and the relational database.

Review Questions

1. What are the different ways to map objects to relational data?
2. How can you insert data into the databases using the LINQ to SQL technology?
3. Why is it easier to use the `SelectNodes` method as compared to the `GetElementsByTagName` method?
4. How do you create XML documents in ASP.NET?
5. What are the advantages of using the `XPathNavigator` class in search?

■ Case Scenario

Scenario 5-1: Storing Information

You want to store information in a particular folder from the user's hard disk in your machine. This includes folders and files. Folders can have nested folders and files. The captured information might be sent to some other components that are developed in different languages and running on different platforms. How will you store this information?



Workplace Ready

Using LINQ

You are developing a web application that generates various reports based on various database tables. After getting the information from the database, you need to validate by comparing it against the business access layer objects that are .NET objects. How will you fetch the information from the database so that your validation becomes easy and fast?

Hint: Use LINQ.

Working with ASP.NET AJAX

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing AJAX.	Implement web forms by using ASP.NET AJAX.	5.1
Using AJAX in ASP.NET applications.	Interact with the ASP.NET AJAX client-side library.	5.2
Using AJAX in ASP.NET applications.	Create and register client script.	5.4
Using AJAX in ASP.NET applications.	Consume services from client scripts.	5.3

KEY TERMS

asynchronous

Asynchronous JavaScript and XML (AJAX)

JavaScript Object Notation (JSON)

postback

serialization

web services

In today's world, most desktop applications are migrating to web applications. Most web applications, which need to transfer data back and forth between the client and the server, post page data to a server and get back a response. It is challenging to create rich user responsive web pages without using full-page refresh. You need to decide whether you want to use complete page postback or partial page postback based on your requirement for a responsive application. Page postback causes complete page rendering, which may increase the page load time. For example, while filling online forms, you may have a long wait while the page refreshes. This is because even small data requests, such as filling in the country and state details or the area code from some type of server control, may cause the entire page to refresh. Also, if you happen to fill in some data incorrectly that requires server-side validation, it may throw a validation error during page postback. AJAX provides a fast and responsive user interface and helps overcome these kinds of challenges.

■ Introducing AJAX



THE BOTTOM LINE

ASP.NET *Asynchronous JavaScript and XML (AJAX)* enables users to communicate between the web browser, which acts as the client, and the server that has the back-end code. To do this, ASP.NET AJAX creates proxies of web services and page methods. This reduces the wait time and provides a better experience to users of the web application.

AJAX is a platform-independent, client-side technology that uses SOAP and XML to send and receive **asynchronous** requests and responses to and from the server. It then uses lightweight technologies, such as JavaScript, DOM, HTML, and CSS to process the response. This lesson describes AJAX controls and its server-side and client-side support. In addition, the lesson covers extender controls and explains how to debug unhandled exceptions in AJAX.

Exploring AJAX

The current generation of server controls is powered by the AJAX script framework. This framework enables the client-based AJAX programming model to work independently.

For example, consider a web application that allows you to fill in a form and then view the bank statements for a specific credit card number. In this scenario, once the form is submitted, the credit card number needs to be validated. On server-based web applications, you can either submit the page once you have filled in all the fields or by doing a page **postback**. However, in the case of a page postback, the entire page may be refreshed. To avoid this, you may need to use multiple variables or sessions to retain the data. This may make your page bulky. In such a scenario, you can use AJAX, which enables you to post the page partially and fetch the data asynchronously without refreshing the entire page.

Using AJAX, you can build web applications with rich user interface and improved response times. You can improve performance of your application by using partial-page rendering and client-side scripting. In addition, it helps your application function like desktop applications by providing a rich user interface.

Understanding How AJAX Works

Most web applications are meant to display data from the server on a remote desktop. Web applications allow users to modify the data remotely and store it back to the server. This also involves submitting user-completed forms to the server. This client-server communication involves Hypertext Transfer Protocol (HTTP). The browsers raise HTTP requests, and the server sends the response back through the HTTP. The AJAX model incorporates a scriptable tool to communicate with the web server using the HTTP. As a result, only the identified sections of the page can be refreshed—the entire page is not refreshed. You can also add controls to the web pages at runtime. This is possible because the page displays content without refreshing the Document Object Model (DOM) of the page with the server-response data. The user experiences uninterrupted data access because the page is not replaced.

The introduction of AJAX has increased the importance of browsers in a web application. Earlier, the web browsers simply rendered the HTML streams and executed the client-side script blocks. AJAX has introduced some new client-script libraries that enable asynchronous callbacks to the server. AJAX also includes some server-side components to support the asynchronous calls raised by the client (browser).

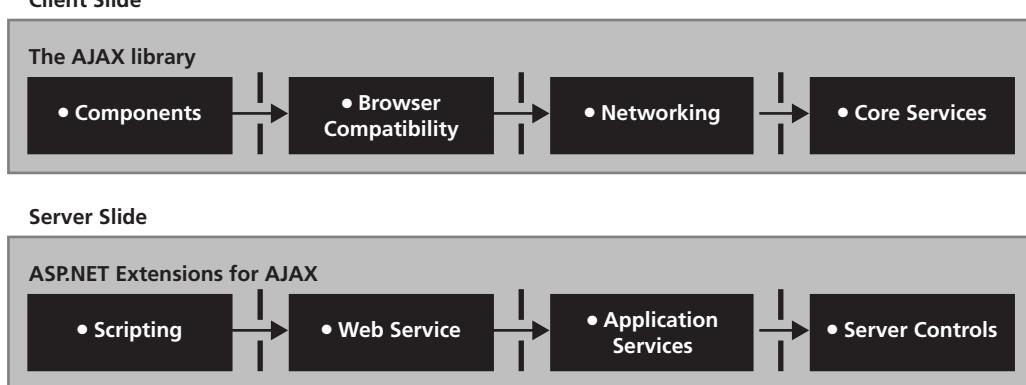
TAKE NOTE *

Postbacks refer to refreshing the entire web page. During page postbacks, the complete page is updated and ViewState is updated too. Callbacks refer to sending a request to the server where the entire page is not refreshed. A callback enables you to fetch only the required data. ViewState is retained on the client end, and the refreshed data may not be updated in the ViewState. Therefore, you have to make sure programmatically that your ViewState is synchronous with the refreshed data. For more information, refer to www.asp.net/ajax.

Figure 6-1 shows the client-side and server-side support of AJAX.

Figure 6-1

AJAX client-side and server-side support



This figure shows the client-side and server-side functionality of AJAX. It shows the client-side support for creating client components, browser compatibility, networking, and core services and the server-side support for scripting, web services, application services, and server controls.

Understanding AJAX Architecture

The architecture of AJAX influences your decisions in designing client and server interaction.

You can achieve an easy and efficient communication channel with AJAX. The alternative—using traditional web application design—will result in slow, difficult handling of data interaction. The communication technique introduced by AJAX is a complete paradigm shift from the way the client and server used to communicate in the standard postback technique. The request raised by the client-side script used to be encountered by the server-side event handler. ASP.NET AJAX is devised to provide this kind of communication with the help of asynchronous calls to the server. It is also devised to eradicate the need for entire page postback calls. Client-side programming has become as powerful as server-side programming in many ways. For example, by using AJAX, the client can also process the received data locally after triggering the request from the browser.

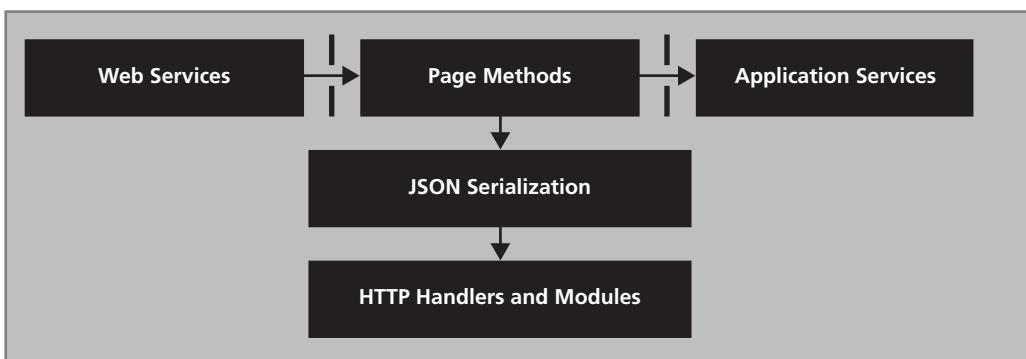
TAKE NOTE*

Powerful client-side programming does not mean that all future applications should be client-centric. Each type of programming has its importance in web application development. However, you can overcome the limitations faced in server-side programming by client-side programming and vice versa. If there is some business logic involved behind the control, it is always advisable to keep it on the server-side for security reasons. If you try to make your application client-centric, then you may have to take extra care in maintaining performance as the page becomes bulky.

It is important to understand how the different layers of the ASP.NET AJAX server communication architecture provide various services. Figure 6-2 shows the ASP.NET AJAX server communication architecture.

Figure 6-2

ASP.NET AJAX server communication architecture



The web services, page methods, and application service methods facilitate communication with the server for various levels of interactions. The HTTP Handler and Modules layer is built on the services of ASP.NET 2.0 to provide support for ASP.NET AJAX. This layer is the backbone support that allows all the communication to happen between the server and the client.

Let's understand these layers in detail.

TAKE NOTE *

REST is an architectural style. Roy Fielding coined the term REST.

USING WEB SERVICES

AJAX provides the ability to call ***web services*** from JavaScript clients with the help of JavaScript Object Notation (JSON). You don't need to rely on Simple-Oriented Application Protocol (SOAP). AJAX also supports Windows Communication Foundation (WCF) where the new enhancements support the Representational State Transfer (REST) web service.

USING PAGE METHODS

Page methods allow ASP.NET pages to call back to the server without passing through the page life cycle or creating a web service. Page methods are a special type of web service that allow you to create a method on an ASP.NET page and expose it to the REST resource. It is a kind of special web service that is automatically created when using page methods. You do not have to create page methods explicitly.

USING SERVICE PROXIES

The service proxy layer in the architecture provides the proxy service for application services, page methods, and the WCFProxy class for these methods. This layer provides classes and methods that can create asynchronous responses to the server. The built-in proxies interact with the ASP.NET application services, such as the role, profile, and authentication services, and have the ability to work with ***JavaScript Object Notation (JSON)***.

USING JSON

JSON is the lightweight data interchange format, which is mainly syntax that represents data the way XML does, but at the same time, is a subset of JavaScript language.

Understanding JSON

JSON allows data-related programming at the client-end, thereby making AJAX more powerful. Using JSON and AJAX, you can use client controls heavily on your page.

JSON is available for the client controls that have to interact with data, making scripting equally powerful, like server-side coding.

Syntax for JSON is based on Object Literal Notation, a method of creating object literals with some subtle differences. In JSON, you add property within single quotes or double quotes. The property can even be without the quotes in JSON. The other difference is that in Object Literal Notation, any primitive type or object can be assigned to a property, while in JSON, only primitive datatypes, arrays, null, and object literals can be assigned to properties.

This is a simple JSON code:

```
var obj =  
{  
    "Name": "ABC",  
    'Address1': "Address line 1",  
    Address2: "Address line 2"  
}
```

Let's see an example of how to use JSON in web applications. Here is a JSON file named Employee.json:

```
Employee.json  
{"Employees": [  
]
```

```
{"Employee":  
    {"ID":"1"},  
    "name":"ABCD",  
    "Address":"Address of ABCD"  
},  
{"Employee":  
    {"ID":"2"},  
    "name":"PQRS",  
    "Address":"Address of PQRS"  
},  
{"Employee":  
    {"ID":"3"},  
    "name":"LMNO",  
    "Address":"Address of LMNO"  
},  
{"Employee":  
    {"ID":"4"},  
    "name":"XYZ",  
    "Address":"Address of XYZ"  
}  
]
```

This file contains standard Employee information like employee id, name, and address. You need to display Employee names in your web application.

Now, create a web page named EmpJSON.aspx. You have an HTML button `btnEmployee` and a `<div>` tag with an ID `zone` section that will display employee names.

Add following code in EmpJSON.aspx file:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="EmpJSON.aspx.cs" Inherits="EmpJSON" %>
<!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>Untitled Page</title>
    <script type="text/javascript" language="javascript">
        function ShowEmp()
        {
            GetJSONFile();
        }
        var fname = "Employee.json";
        function GetJSONFile()
        {
            var xhr = createXHR();
            xhr.open("GET", fname,true);
            xhr.onreadystatechange=function()
            {
                if (xhr.readyState == 4)
                    { if (xhr.status != 404)
                        { document.getElementById("zone").innerHTML = "found";
                            ShowEmployee(xhr.responseText);
                        }
                    }
                else
                    { document.getElementById("zone").
```

```
innerHTML = fname + " not found";
        }
    }
}

xhr.send(null);
}

function ShowEmployee(jsoncontent)
{
    var data = eval("(" + jsoncontent + ")");
    document.getElementById("zone").innerHTML = "Emp Name";
    document.getElementById("zone").innerHTML += "<br>";
    for(i = 0; i < 4; i++)
    {
        var line = " ";
        line += "<br>" + data.Employees[i].name;
        document.getElementById("zone").innerHTML += line;
    }
}

function createXHR()
{
    var request = false;
    try
    {
        request = new ActiveXObject('Msxml2.XMLHTTP');
    }
    catch (err2)
    {
        try
        {
            request = new ActiveXObject('Microsoft.XMLHTTP');
        }
        catch (err3)
        {
            try
            {
                request = new XMLHttpRequest();
            }
            catch (err1)
            {
                request = false;
            }
        }
    }
    return request;
}
</script>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <input id="btnEmployee" type="button" onclick="ShowEmp();"
value="Get Employee" />
            <br />
            <br />
            <div id="zone">
            </div>
        </div>
    </form>
</body>
```

```

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

```

On clicking the Get Employee button, the ShowEmp() function is called, which calls another function GetJSONFile(). The GetJSONFile will open the Employee.json file and read the JSON file. It will then use the ShowEmployee () function to get Employee names and finally display them in the <div> tag.

JSON can be used in two ways:

- **As a name-value pair:** In the name:value format, it can be used within the left and right brackets separated by commas { , }.
- **As an array:** It can be used as an array with comma-separated data inside the left and right [] brackets.

You can use the methods provided by the JSON object to convert values to and from the JSON format. The data transfer to and from JSON is carried out by the **serialization** layer. Serialization is converting data from one format to another. Data exchange involve three steps: the browser processing, the server processing, and the data exchange between them. You can use the `JSON.stringify` method to serialize a JScript value to JSON and `JSON.parse` method to deserialize a value from the JSON to the JScript format.

MORE INFORMATION

For more information about Object Literal Notation and JSON, refer to json.org.

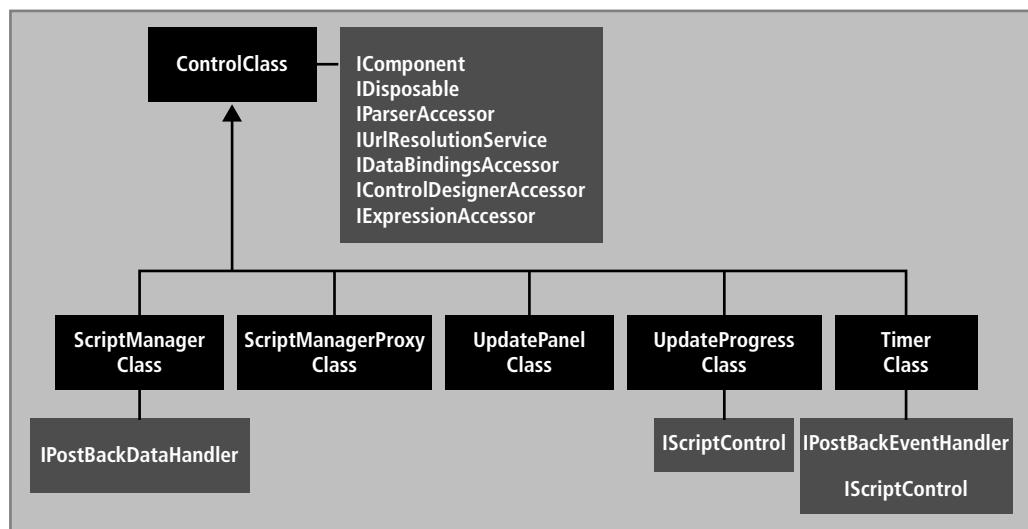
Working with ASP.NET Server-Side AJAX Support

The ASP.NET AJAX toolkit comes with server support by providing server-side controls that are meant to be displayed on the browser.

These controls are simple to use; and like any other control, you can drag and drop them on your design screen. AJAX controls provide partial page rendering or updating and client-to-server progress updating. The process of ASP.NET page rendering that is used to render the entire bulky page in one go is broken down into smaller bits. Figure 6-3 shows the classes involved in rendering these smaller chunks.

Figure 6-3

Classes involved in ASP.NET Extension controls



`System.Web.UI.Control` is the core control that takes care of server-side support for AJAX. This server control correctly renders the output as the HTML that will appear on the browser. Suppose you want to render a `ListBox` control. It is displayed on the HTML screen as a `<select>` tag, but the `TextBox` control is represented as an `<input type= "text"/>` tag.

Some frequently used ASP.NET AJAX controls are `ScriptManager`, `ScriptManagerProxy`, `UpdatePanel`, `UpdateProgress`, and `Timer`. The next lesson discusses these controls.

Working with ASP.NET Client-Side AJAX Support

If you have written code in older versions of ASP.NET that are not AJAX-rich, then you will easily recollect that the only way to save the page from postback and avoid page rendering is to write client-side script.

These are the component layers that contribute to ASP.NET AJAX script libraries:

- **Browser compatibility:** Provides compatibility with most widely used browsers, such as Internet Explorer, Mozilla Firefox, and Apple Safari. Built-in support is available for the server controls. Note that this compatibility is for the client end.
- **ASP.NET core service layer:** Extends the basic JavaScript environment by including classes, namespaces, event handling, datatypes, and object serialization.
- **ASP.NET AJAX base class library:** Involves components for string management and extended error handling.
- **AJAX networking layer:** Manages the communication with web-based services and applications. It also handles asynchronous remote method calls. This is a client-side support.

CERTIFICATION READY?

Implement web forms
by using ASP.NET
AJAX.
5.1

■ Using AJAX in ASP.NET Applications



THE BOTTOM LINE

AJAX provides several built-in server-side and client-side controls. Let's learn about these.

Working with AJAX Server-Side Controls

You can use the various server-side controls provided by AJAX to design interactive web forms.

USING THE SCRIPTMANAGER AND SCRIPTMANAGERPROXY CONTROLS

The ScriptManager control pushes the AJAX library script to the client page at page request. A single ASP page is allowed to hold only one instance of the ScriptManager control. If the default setting of the `EnablePartialRendering` property is set to `true`, then the ScriptManager control makes partial page rendering possible. It also supports localization as well as custom user scripts. ScriptManager is not a visual control on the page; instead, it is only meant to manage processing of AJAX and other controls.

The code for the ScriptManager control in the page appears like this:

```
<asp: ScriptManager ID="ScriptManager1" runat="server">  
</asp: ScriptManager>
```

The ScriptManagerProxy control is required to render nested controls, such as content pages and user controls. It manages the service references of the pages that already have a ScriptManager control.

TAKE NOTE *

The master page applied on a content page is not treated as a separate page.

Usually in the case of the master page, if there is already a ScriptManager control present and the content page using the master page requires a ScriptManager, introducing another ScriptManager control causes an error. There can be only one instance of the ScriptManager control. In such cases, you can use the ScriptManagerProxy control.

USING THE UPDATEPANEL CONTROL

The UpdatePanel control is used to define the scope for partial rendering, that is, postback to the server without depending on the entire page postback event. Partial page rendering means that the section within the UpdatePanel control is posted, and data pertaining to that section is retrieved and refreshed on the client screen. This saves on the entire page processing. The

difference can be observed at the client end, when the data appears on the screen as if it is being processed at the client end. Technically, it's a container for other controls that need partial rendering.

It is important to understand how UpdatePanel behaves when the page postback occurs and how partial rendering is handled on the client and server ends. When a server control is placed in the UpdatePanel control, then the page rendering can be performed only for the section of the page that lies within the UpdatePanel control.

To partially render a page, the client triggers the postback and carries out the following steps:

1. Triggers partial rendering.
2. Composes the request.
3. Sends the request to the server.

The server takes appropriate action for the request as follows:

1. Loads the page.
2. Loads the controls.
3. Renders only those controls that are within the UpdatePanel control.
4. Renders the page.
5. Sends the response back as a partial postback.

The client further processes the request by:

1. Accepting the response from the server.
2. Destroying the already existing DOM elements contained within an updating UpdatePanel.
3. Replacing the contents of UpdatePanel with the received content.
4. Processing the scripts, if any.

The postback event is triggered by the UpdatePanel control and sent to the server for processing. The server handles the request as any other page postback request by executing the page life cycle as usual, and then sends the response back to the client after rendering the controls to `HtmlTextWriter`.

This example shows you how server-side controls allow you to perform partial page updates. In this example, a web page with two labels is loaded. One label is outside UpdatePanel and the other label is inside UpdatePanel. The UpdatePanel control supports a collection of triggers that will generate partial page updates. Triggers are collections of control events that may trigger a postback and refresh UpdatePanel. You will see how to use triggers in the UpdatePanel control section.

The default.aspx file contains the code to demonstrate partial rendering, as shown in the following code example:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
```

```
<ContentTemplate>
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    <br />
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Partial Postback" />
    <br />
    <br />
</ContentTemplate>
</asp:UpdatePanel>

</div>
    <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
    <br />
    <asp:Button ID="Button2" runat="server" Text="Page Postback" />
</form>
</body>
</html>
```

The default.cs file contains the following code:

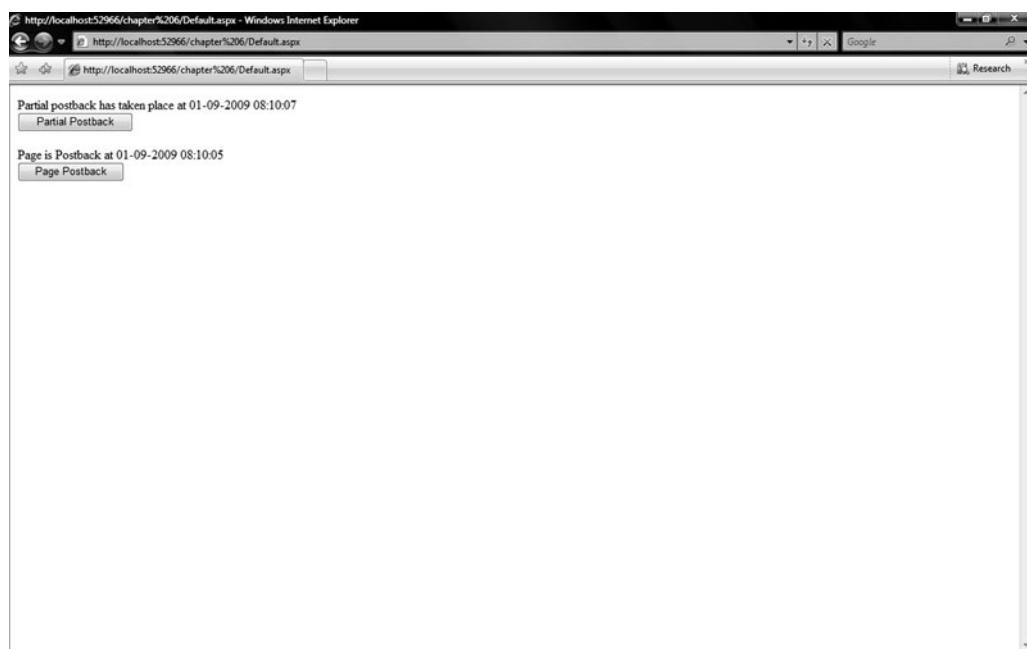
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = "Page is postback " +
Convert.ToString(DateTime.Now);
        Label2.Text = "Page is postback " +
Convert.ToString(DateTime.Now);
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Partial postback has taken place at " +
Convert.ToString(DateTime.Now);
    }
    protected void Button2_Click(object sender, EventArgs e)
    {
        // This will cause the complete page to load as it is
placed outside the UpdatePanel1.
    }
}
```

In this code, the page contains two labels, one inside the UpdatePanel control and another outside the UpdatePanel control. It also contains two buttons in similar fashion, one within the UpdatePanel control and another outside. This code places a ScriptManager control before placing the UpdatePanel control to the page. The ScriptManager is required on the page to execute the AJAX controls. When the page is loaded for the first time, both labels display the message “Page is postback at 01-09-2009 08:10:05.” If the Partial Postback button is clicked, only the text of label1 will change, that is, within the UpdatePanel to “Partial postback has taken place at 01-09-2009 08: 10:07.” This demonstrates that the label outside the UpdatePanel1 has not been refreshed, and partial rendering has taken place. Figure 6-4 shows the output of the code.

Figure 6-4

Output of using the UpdatePanel control



By default the UpdatePanel control is refreshed if any of its child control causes postback. The behavior of the UpdatePanel control can be altered with the help of the public properties such as Trigger, ChildAsTrigger, and UpdateMode.

Trigger is mainly a collection of control events that will trigger the postback and refresh the UpdatePanel control. From the previous example, the Button control triggers the postback, therefore, you can write the code as:

```
<asp:UpdatePanel ID="UpdatePanel12" runat="server"
UpdateMode="Conditional">
    <ContentTemplate>
        Update Panel Two [Conditional updates]<br />
    &nbsp;
        <asp:Label ID="Label3" runat="server"></asp:Label>
        <br />
        <br />
        <asp:Button ID="Button2" runat="server" Text="Update Panel Two"
Width="255px" />
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Button2" EventName="Click" />
    </Triggers>
</asp:UpdatePanel>
```

In the earlier code, the Button control will trigger the server event and refresh the UpdatePanel control. Using the PostBackTrigger or the AsyncPostBackTrigger tags, you can create the collection of events. For example, here you have added the Click event of the Button control.

The **ChildAsTriggers** property of the UpdatePanel control is **true** by default. If the page cannot be easily split into panels, then the child section of the page, which should trigger postback, can be set to **false**.

Here is a code sample that uses **ChildAsTriggers**. In this case, the code can be written as:

```
<asp:UpdatePanel ID="UpdatePanel11" runat="server"
ChildAsTriggers="true">
```

```
<ContentTemplate>
    <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
    <br />
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
Text="Partial PostBack" />
    <br />
    <br />
</ContentTemplate>
</asp:UpdatePanel>
```

TAKE NOTE *

Remember that the conditional setting works only if you have multiple panels on the page.

In this case, the postback will be triggered by the Click event of the Button control, but no markup will be generated that will refresh the panel.

The UpdateMode property is set to refresh unconditionally by default. However, the UpdatePanel control can be set to refresh under some condition by setting it to Conditional, as follows:

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server"
UpdateMode="Conditional">
```

IMPLEMENTING PARTIAL PAGE RENDERING USING THE UPDATEPANEL CONTROL

You use partial page rendering to update different parts of a page. For example, say you have a web page where you need to display stock updates that get refreshed periodically. You also have a section on the page where you can select a company and view the details of mutual funds of that company. In such a scenario, you can use multiple UpdatePanels. Each of the UpdatePanels will work independently on the same page. This is known as partial page rendering.

The UpdatePanel has a property, UpdateMode, which you can set either Always or Conditional. The setting Always specifies that the UpdatePanel will be refreshed on every page postback, while the Conditional setting specifies that the UpdatePanel will be refreshed when it meets specified conditions. You can explicitly call the Update method to do a conditional refresh of the UpdatePanel.

USING THE UPDATEPROGRESS CONTROL

The UpdateProgress control handles intermediate feedback for the operations that have already been running for a while. For example, it provides status information about the partial-page updates that occur within the UpdatePanel control. The information may be displayed in an animated.gif file to show that the system is busy processing. Users may be asked to wait while the partial-page update completes.

The UpdateProgress control has three properties:

- **AssociatedUpdatePanelID:** Refers to the ID of the UpdatePanel control that is bound to the UpdateProgress control.
- **DisplayAfter:** Defines the time lag before the UpdateProgress control displays content. The default is 500 milliseconds (half a second). This means that if the callback completes faster than the specified time, the progress control does not need to be shown. The DisplayAfter property is used for implementing this functionality.
- **DynamicLayout:** When the DynamicLayout property is true, the UpdateProgress control can be displayed dynamically. The <div> tag that holds the control should set the display property to None. The page dynamically changes the layout and allocates the space at random. When the DynamicLayout property is false, the <div> tag that holds the control sets the display property to block and the visibility property to hidden. By doing this, the space for the Progress control will be allocated when the page is initially rendered.

The following code shows you how to use the UpdateProgress control:

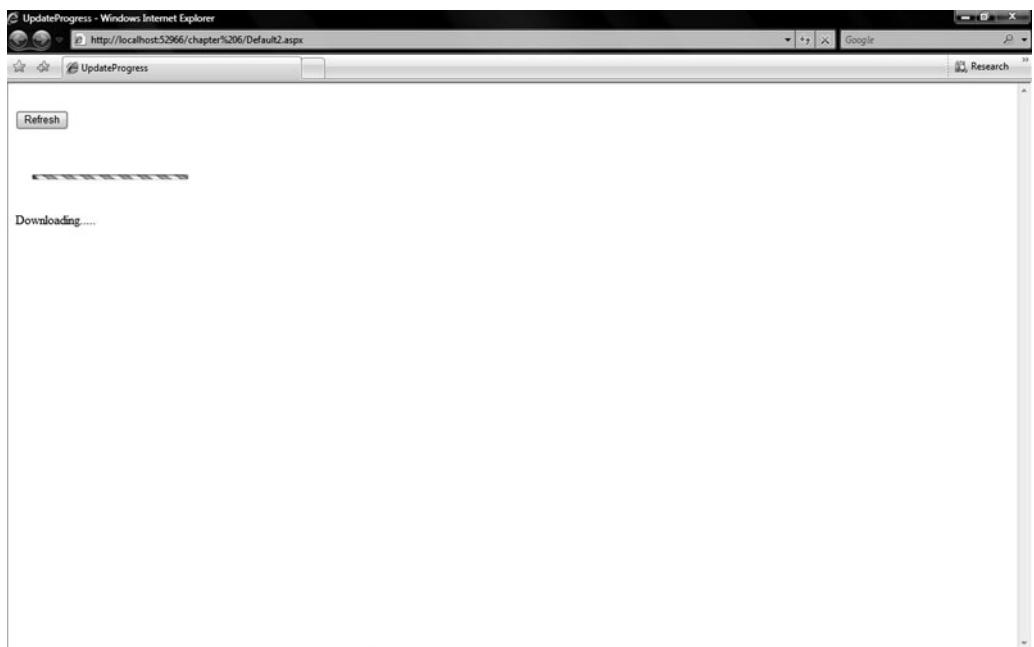
```
<% @Page Language="C#" AutoEventWireup="true"
CodeFile="Default2.aspx.cs" Inherits="Default2" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(5000);
    }
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>UpdateProgress</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <br />
                    <asp:Button ID="Button1" runat="server"
onclick="btnSubmit_Click" Text="Refresh" />
                </ContentTemplate>
                </asp:UpdatePanel>
                <br />
                <asp:UpdateProgress ID="UpdateProgress1" runat="server"
AssociatedUpdatePanelID= "UpdatePanel1">
                    <ProgressTemplate>
                        <asp:Image ID="Image1" runat="server" Height="80px"
                            ImageUrl="~/Images/image1.jpeg" Width="236px"
Visible="True" />
                        <br />Downloading.....
                    </ProgressTemplate>
                </asp:UpdateProgress>
                <br />
                <br />
            </div>
        </form>
    </body>
</html>
```

This code uses the ScriptManager control, the UpdatePanel control, and a button that is placed within the UpdatePanel1 control. The progress image is placed within the UpdateProgress control. To give a time delay, the `btnSubmit_Click` event has a time lag of 500 msec using C# and is called on the click of Button1. Executing the page shows an output. A screen shot of this output is shown in Figure 6-5.

Figure 6-5

Output of using the UpdateProgress control



USING THE TIMER CONTROL

The Timer control allows you to schedule the postback at a defined interval in a periodic manner. The information that needs to be refreshed on the screen at a regular interval can be driven by the Timer control. For example, reservation availability, stock market data, or a simple sequence of images loading one after the other can be driven by the Timer control.

The Timer control can be used directly on the UpdatePanel control. Based on the time defined, the automatic partial-page update will be triggered.

The following code shows you how to use the Timer control:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default3.aspx.cs" Inherits="Default3" %>

<!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>
    </title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ScriptManager ID="ScriptManager1" runat="server">
            </asp:ScriptManager>
            <asp:Timer ID="Timer1" runat="server" Interval="2000">
            </asp:Timer>
            <asp:UpdatePanel ID="UpdatePanel1" runat="server">
                <ContentTemplate>
                    <asp:Label ID="Label1" runat="server" Text="Keep Watching,
the page will refresh after every 2 sec">
                    </asp:Label>
                </ContentTemplate>
            </asp:UpdatePanel>
            <br />
        </div>
    </form>
</body>
</html>
```

CERTIFICATION READY?

Interact with the ASP.NET AJAX client-side library.

5.2

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

This code adds an UpdatePanel control and a Timer control. Time is set for an interval of two seconds. A Label control is placed in the UpdatePanel control to display a message. The page will refresh after every two seconds. The output will refresh after every two seconds.

Working with AJAX at Client Side

Page postback is a common way to transmit data between the client and server in an ASP.NET application. Though this technique is simple, it is quite inefficient, especially when you want to update a single value in a web page. One workaround for this problem is to use client-side controls.

For example, if validation is required on a text box for the length of the text or a text format, it is advisable to validate the control at the client side and prompt a warning message then and there without hitting the server. Using client-side controls eliminates delays in loading a web page. The web page does not need to load all over again for minor changes. In this manner, AJAX makes client scripts more powerful.

There are three ways to use the client script in your ASP pages:

- By defining the script block
- By dynamically adding the script
- By using the ScriptManager control

DEFINING THE SCRIPT BLOCK

You can add the script block straight to the code using the include attribute by referring to the .js files. Even though the script block facilitates client-side scripting, it does not utilize the AJAX library feature. This is the traditional way of writing client script.

Let's consider a simple example. Suppose, a user needs to select a state from the drop-down list named ddlStates. As the user selects the country, based on the selection, the corresponding states for that country should be displayed in the drop-down list. Initially, the ddlStates drop-down will be inactive. The states list will be populated only after selecting a valid country.

You can see here how the script block appears in a .aspx page. Here is a simple code for btnSubmit_Click. You can call this function on the OnClick event of a button:

```
<asp:Button ID="Button1" runat="server" onclick="btnSubmit_Click"
Text="Refresh" />
<script runat=server>
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        System.Threading.Thread.Sleep(5000);
    }
</script>
```

DYNAMICALLY ADDING THE SCRIPT

Depending on the server-side processing, the ClientScriptManager can dynamically add javascript at runtime.

In addition to defining the script block, you can generate the client script at runtime and attach it to the ASP.NET pages. The ClientScriptManager class is used to register Javascript to the page. The Page object's client script property exposes the ClientScriptManager class. This property adds the JavaScript to the page at runtime.

Adding JavaScript may be required in a scenario where the controls are generated on the page at runtime. You can add JavaScript to the page by calling the ClientScriptManager object's registerClientScript-block method.

Here is a simple script code to demonstrate the use of ClientScriptManager and RegisterClientScript:

```
<%@ Page Language="C#"%
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        // Obtain the ClientScript property from the Page object.
        ClientScriptManager objClientScriptManager =
this.Page.ClientScript;
        // Check whether the script block is already registered
        if
(!objClientScriptManager.IsClientScriptBlockRegistered(this.GetType(), "Alert"))
        {
            objClientScriptManager.RegisterClientScriptBlock(this.GetType(), "Alert", "alert('This is example of ClientScriptManager!')", true);
        }
    }
</script>
<html>
    <head>
        <title>ClientScriptManager</title>
    </head>
    <body>
        <form id="Form1"
            runat="server">
            Example of ClientScriptManager
        </form>
    </body>
</html>
```

The code displays an alert on the page load event.

Another example of dynamically adding script can be a scenario where the user is registering on a site and needs to first satisfy an age criteria. The check box on the form should be checked if the user's age is more than 18 years. The occupational detail drop-down list will appear only if the user's age is more than 18, and the JavaScript will be executed dynamically to ensure that the user selects the age limit in the CheckBox.

USING THE SCRIPTMANAGER CONTROL

Another way of client-side scripting is by using the ScriptManager control, which is a server control. This control registers JavaScript with the ASP.NET page.

The ScriptManager control is an AJAX extension server control as mentioned earlier, and it automatically registers the appropriate script files defined by the AJAX library. It also allows registering your own script with the page, which can be done declaratively or programmatically. The <Scripts> collection element of the ScriptManager control is used for this purpose.

You can use the ScriptManager control to register a script (.js) file that is embedded in the assembly or present on a Web site. To do so, you need to use the ScriptReference object. You can set the following properties on this object:

- **Path:** Specifies the location of the static script file.
- **Assembly:** Specifies the name of the assembly that contains the script file that is embedded in the assembly.
- **Name:** Specifies the name of the .js file that is embedded in the assembly.
- **ScriptMode:** Specifies whether to use the debug or release version of the script.

CERTIFICATION READY?

Create and register client script.

5.4

Consider this code snippet:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <scripts>
    <asp:ScriptReference Path("~/timer.js" />
  </scripts>
</asp:ScriptManager>
```

This code explains how to use the **ScriptReference** object to call the script file. In this case, the script file is Timer.js.

Working with AJAX Events

An AJAX web page raises two kinds of events: client events and server events.

You can use client events to manage custom script components and customize UI for postback operations. To raise client events, you use the client classes provided by the AJAX library. Note that these classes are added to the web page whenever you use AJAX server controls on the page.

Using client events, you can cancel postbacks, prioritize postbacks, and refresh the content in **UpdatePanel** controls.

The **Application** and **PageRequestManager** classes are the primary classes that raise events during the client life cycle of an ASP.NET AJAX page.

USING APPLICATION CLASS EVENTS

The client events of the **Application** class are:

- **Sys.Application.init**: This event is raised when you load scripts and render a page.
- **Sys.Application.load**: This event is raised for all postbacks to the server when you initialize all objects in an application.
- **Sys.Application.unload**: This event is raised prior to terminating all objects in the application.
- **Sys.Component.propertyChanged**: This event is raised when a property of a component changes.
- **Sys.Component.disposing**: This event is raised when the object of the **Application** class is terminated.

USING PAGEREQUESTMANAGER CLASS EVENTS

The client events of the **PageRequestManager** class are:

- **Sys.WebForms.PageRequestManager initializeRequest**: This event is raised when the instance of the **Application** class is terminated.
- **Sys.WebForms.PageRequestManager beginRequest**: This event is raised before an asynchronous postback is sent to the server.
- **Sys.WebForms.PageRequestManager pageLoading**: This event is raised after the server sends the response for an asynchronous postback, but before the page is updated with the new content.
- **Sys.WebForms.PageRequestManager pageLoaded**: This event is raised after all content on the page is refreshed due to the postback operation.
- **Sys.WebForms.PageRequestManager endRequest**: This event is raised after the server processes the response for an asynchronous postback and updates the page. This event is also raised if there is an error while processing the response. The page is not updated in case of an error.

Server controls have their own events similar to the events in the life cycle of an ASP.NET page. During its life cycle, an ASP.NET page goes through a number of stages, such as initialization, loading, instantiating controls, event handling, and rendering. During these stages,

it raises a number of events. You can use event handlers to handle these events and run the code. Some typical page events are:

- PreInit
- Init
- PreLoad
- Load
- PreRender
- Unload

You can use these events in JavaScript objects with multiple handlers. You can also use these events with multiple functions. You can expose these events by creating methods that add or remove an event handler. You can also create a method that raises the event.

For example, you can add and remove the `Sys.Application.init` event as shown in the following code:

```
Sys.Application.add_init(PageInitHandler);  
  
function PageInitHandler() {  
// Add code to handle the event  
}  
Sys.Application.remove_init(PageInitHandler);
```

X REF

For more information on events, refer to Lesson 1, “Introducing ASP.NET 3.5.”

The ASP.NET AJAX Control Toolkit from Microsoft is a community project made up of Software Development Kit (SDK) and code samples. The toolkit provides controls that you can use in your applications to make them rich. You can use the ASP.NET AJAX Control Toolkit as a foundation to build your own controls, like user/custom controls. The components are available to you for development depending on the options you specify during installation.

The Ajax Control Toolkit is available in two versions: stand alone and complete. The stand-alone version is best suited for development of simple controls. It also contains a sample test Web site. To install the stand-alone version, extract the zip file to a directory and run the `AjaxControlExtender.vsi` file located in the `AjaxControlExtender` subdirectory. The Complete version allows you to create, test, and debug your own controls. It includes source code, the Toolkit library project, a sample Web site project, a unit test harness project, and a template project. The Toolkit contains rich Calendar control, ComboBox, DragPanel, ListSearch, and others. You can download the Toolkit and AJAX toolkit samples from the following URL: <http://www.asp.net/ajax/AjaxControlToolkit/Samples/>.

In addition to the AJAX controls mentioned so far, you can use extender controls to enhance the functionality of the AJAX controls. The extender controls are lightweight because they work with the ASP.NET page HTML code, unlike the basic controls such as ScriptManager or Timer control, which add a lot of script code on rendering the page. Examples of extender controls are AutoComplete Extender and Modal Popup Dialog style Extender.

To learn more about these extender controls, refer to MSDN.

TAKE NOTE*

Understanding Web Services in AJAX

You can use ASP.NET web services in your AJAX applications. You can use HTTP requests to call the methods of these web services. Let's learn how to use client script in an AJAX web application and call a web service.

Web services can be ASP.NET web services (.asmx services), or Windows Communication Foundation (WCF) services (.svc services). You can create and use ASP.NET web services and access them from client scripts in your AJAX applications. Often, web services in AJAX are local methods and are called within the same application. These web services exchange data using JSON.

CALLING WEB SERVICES USING WEBMETHODS

A common way of calling web services is by using page methods. Page methods are static methods that you define on an ASP.NET page. These page methods are known as WebMethods.



CREATE A WEB SERVICE

Let's create a web service in AJAX:

1. Create a new Web site.
2. Click Add > New Item, and then, select Web Service. Name it "HelloWorld.asmx."

We'll use the default Hello World WebMethod that is created within the file. Note that when you add the web service, two files are created within the App_code directory: HelloWorld.ascx and HelloWorld.cs.

3. Open the HelloWorld.cs file from the App_Code folder in the Solution window and copy the following code. Note that when you use your own code, you need to add the namespace that you created earlier.
4. Add the namespace Samples.AspNet, as shown in the following code:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

namespace Samples.AspNet
{
    [WebService(Namespace = "http://mycompany.org")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    // To allow this web service to be called from script, using
    ASP.NET AJAX, uncomment the following line.
    [System.Web.Script.Services.ScriptService]
    public class HelloWorld: System.Web.Services.WebService
    {
        public HelloWorld()
        {
            // Uncomment the following line if using designed components
            // InitializeComponent();
        }

        [WebMethod]
        public string HelloWorldMethod()
        {
            return "Hello World";
        }
    }
}
```

5. Create a HelloWorld.js file and add the following code to it:

```
var helloWorldProxy;
function pageLoad() {
    helloWorldProxy = new Samples.AspNet.HelloWorld();
    helloWorldProxy.set_defaultSucceededCallback(SucceededCallback);
    helloWorldProxy.set_defaultFailedCallback(FailedCallback);
}
function OnClickHello() {
    var greetings = helloWorldProxy.HelloWorldMethod();
}
function SucceededCallback(result) {
```

```
        var Rs1tElem = document.getElementById("Results");
        Rs1tElem.innerHTML = result;
    }
    function FailedCallback(error, userContext, methodName) {
        if (error !== null) {
            var Rs1tElem = document.getElementById("Results");
            Rs1tElem.innerHTML = "An error occurred: " +
                error.get_message();
        }
    }
    if (typeof (Sys) !== "undefined")
        Sys.Application.notifyScriptLoaded();
```

6. Replace the code in your HelloWorld.asmx file with the following code:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/HelloWorld.cs" Class="Samples.AspNet.HelloWorld" %>
```

7. Now call the method from client script. Add a web form to the solution by clicking Website > Add New Item.

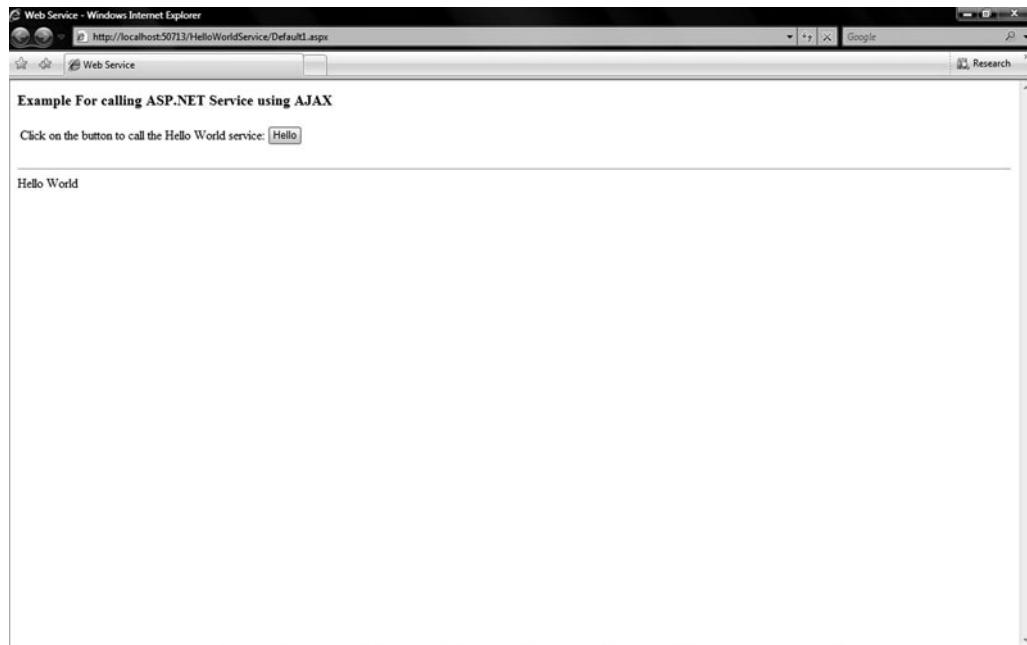
8. Then, copy this code to the default1.aspx file:

```
<%@ Page Language="C#"%
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head id="Head1" runat="server">
        <title>AJAX Service</title>
    </head>
    <body>
        <form id="Form1" runat="server">
            <asp:ScriptManager runat="server" ID="scriptManager">
                <Services>
                    <asp:ServiceReference path="~/HelloWorld.asmx" />
                </Services>
                <Scripts>
                    <asp:ScriptReference Path="~/HelloWorld.js" />
                </Scripts>
            </asp:ScriptManager>
            <div>
                <h3>Example For calling ASP.NET Service using AJAX</h3>
                <table>
                    <tr align="left">
                        <td>Click on the button to call the Hello World service:</td>
                        <td>
                            <button id="Button1"
                                onclick="OnClickHello(); return false;">Hello</button>
                        </td>
                    </tr>
                </table>
            </div>
        </form>
        <hr/>
        <div>
            <span id="Results"></span>
        </div>
    </body>
</html>
```

You can see the result by setting your Default1.aspx file as Set As Start Page and then pressing F5. Alternatively, you can type the URL of your site on Internet Explorer (IE). On clicking the Hello button, the Hello World message will be displayed, as shown in Figure 6-6.

Figure 6-6

Result page

**X REF**

For more information on ASP.NET web services, refer to Lesson 10.

CERTIFICATION READY?

Consume services from client scripts.

5.3

CALL SERVER-SIDE CODE FROM THE CLIENT SIDE USING THE AJAX WEBMETHOD ATTRIBUTE

To use the AJAX `WebMethod` attribute and call server-side code from the client side:

1. Create a Web site by selecting New > Website > ASP.NET Website from Visual Studio. Name the project and save it in the desired location or save it at a default location.
2. Create a `GetProductId` method to get the product id that accepts the product name as the parameter:

```
public string GetProductId(string ProductName)
{
    switch (ProductName)
    {
        case "Cola":
            return("PID00002");
        case "Burger":
            return("PID00342");
        case "Chocolate":
            return("PID122234");
        default:
            return("Product not found");
    }
}
```

In this code, you are passing static data for three products. In real-time applications, you can fetch the data from the database and pass the actual product id, and display the product information.

3. Now, transform this page method into a web method that calls the GetProductId method from the client code using JavaScript. To do this, add the web method along with the GetProductId function:

```
[System.Web.Services.WebMethod]
```

```
public string GetProductId(string ProductName)
{
    switch (ProductName)
    {
        case "Cola":
            return("PID00002");
        case "Burger":
            return("PID00342");
        case "Chocolate":
            return("PID122234");
        default:
            return("Product not found");
    }
}
```

4. Add the attribute EnablePageMethod="true" to the ScriptManager:

```
<asp:ScriptManager ID="ScriptManager1" EnablePageMethod="true"
runat="server">
</asp:ScriptManager>
```

5. Call the GetProductId method from the client side:

```
<body>
<script type="text/javascript" language="javascript" >
    Function callGetProdId(string PdName)
    {
        string PdId = PageMethods.GetProductId(PdName);
        return PdId;
    }
</script>
<form id="form1" runat="server">
```

6. Invoke the method, and whenever the text box loses the focus, add the following code in the Page_Load () event of the page:

```
if (!Page.IsPostBack)
{
    txtId1.Attributes.Add("onblur",
"javascript:CallMe('" + txtId1.ProdID + "', '" + txtProdName1.
ProdID + "')");

    txtId2.Attributes.Add("onblur",
"javascript:CallMe('" + txtId2.ProdID + "', '" + txtProdName2.
ProdID + "')");
}
```

SKILL SUMMARY

In this lesson, you have learned that AJAX allows users to communicate between the web browser and the server using proxies of web services and page methods. You learned that the AJAX script framework, which provides a rich AJAX Control ToolKit, allows the client-based AJAX programming model to work independently. Using the framework does not cause page postback and allows asynchronous server trips without refreshing the client pages.

(continued)

SKILL SUMMARY (continued)

You also learned that the ASP.NET AJAX Toolkit comes with server support by providing server-side controls. It provides client side support by incorporating JavaScript on the client side.

You learned to design interactive web pages using various server-side controls of AJAX and to use client scripts in ASP pages. You also learned how to use extender controls to enhance the functionality of the AJAX controls.

For the certification examination:

- Implement web forms by using ASP.NET AJAX client-side and server-side controls.
- Interact with the ASP.NET AJAX client-side library.
- Create and use web services from client scripts.
- Create, register, and manage client scripts.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. AJAX has introduced new _____ that enables asynchronous callbacks to the server.
2. AJAX provides the ability to call web services from JavaScript clients with the help of _____.
3. _____ is the process of converting a value from JSON to JScript format.
4. The _____ control pushes the AJAX library script to the client page at the time of page request.
5. The _____ control is used to define the scope for partial rendering.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|----------|--|
| T | F 1. The UpdatePanel control handles intermediate feedback for the operations that have been running for a long time. |
| T | F 2. A Timer control cannot be used directly on the UpdatePanel control. |
| T | F 3. The ClientScriptManager class is used to register Javascript in a web page. |
| T | F 4. In JSON, you enclose property names with a single quote mark. |
| T | F 5. A single ASP page is allowed to hold only one instance of the ScriptManager control. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. Which of the following has AJAX included to support asynchronous calls raised by the client?
 - a. Server-side components
 - b. Client-side components
 - c. Client-side libraries
 - d. Namespaces

2. Which of the following controls help you update only a specific part of a web page?
 - a. Timer
 - b. UpdatePanel
 - c. ScriptManagerProxy
 - d. Button
3. You have developed a web page where you have used UpdatePanels. You have placed UpdatePanel B within UpdatePanel A. Now, when you postback UpdatePanel B, which of the following will take place?
 - a. Only UpdatePanel B will get updated.
 - b. Both UpdatePanel A and B will be updated.
 - c. Only UpdatePanel A will be updated.
 - d. None of the UpdatePanels will be updated.
4. Which of the following components of AJAX enables serialization and deserialization of data?
 - a. Service proxy
 - b. JSON
 - c. Page methods
 - d. Web services

Review Questions

1. What is partial page rendering?
2. What are the component layers that contribute toward AJAX script libraries?

■ Case Scenarios

Scenario 6-1: Creating a Registration Page

You have to develop a Web site for your organization. The Web site requires a registration page that will display a set of common fields that a user first needs to fill in and a section with vendor and visitor specific fields. The user should be able to access the relevant fields by selecting the radio buttons Vendor and Visitor. The questions should be loaded dynamically without refreshing the page. How will you achieve this?

Scenario 6-2: Displaying Animations

Consider that you are developing a Web site to provide information about some products. You need to show an animated slideshow of images with the product information on the home page of the Web site. The products will be displayed without any user interaction. However, you should ensure that it is possible to add or delete product information without much hassle later on. Therefore, you cannot use a.gif animation file. How will you develop the Web site?



Workplace Ready

Developing Rich Web Sites

You need to develop a Web site that has a section to display the latest stocks and news updates. These updates should refresh every three minutes. How will you design the Web site?

Programming Web Applications

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing Input Validation.	Implement client-side validation and server-side validation.	2.4
Performing Site Management.	Configure providers.	1.1
Performing Site Management.	Implement data-bound controls.	2.1
Performing Site Management.	Implement session state, view state, control state, cookies, cache, or application state.	7.5
Implementing State Management.	Configure session state by using Microsoft SQL Server, State Server, or InProc.	1.4
Implementing State Management.	Implement session state, view state, control state, cookies, cache, or application state.	7.5

KEY TERMS

input validation
remapping

site navigation
state management

When you program a web application, you can classify it into one of three broad categories: business logic programming, UI-related programming, or data handling-related programming. Among these three categories, UI-related programming acts as the interface between the business logic and the data handling-related programming.

Introducing Input Validation



THE BOTTOM LINE

When a user enters data on a web page, that data needs to be sent to code handling, business logic, and/or the code that saves data in a database.

In web application development, three important concepts in UI-related programming are:

- **Handling input validation:** Making sure that the data input by a user is as needed by the application.
- **Managing navigation across the Web site:** Giving users ways to easily find out where they are and where else they can navigate to.
- **Maintaining the state of the user:** Keeping a persistent data state for pages or applications across multiple trips to and from the server as the user browses across the different pages of the Web site.

In this lesson, you will learn about these three crucial web application programming concepts.

Implementing Input Validation

The data input to any application must be valid for the application, which means that it should be correct. To ensure that the data entered into an application is valid, the input needs to be validated by the application.

The process of checking whether all external input data is valid is called ***input validation***. Examples of input validation in a web application are:

- **Checking the format of data, such as dates:** For example, if an online banking application accepts the date in mm/dd/yyyy format, the entered date must be validated so that it is interpreted and processed correctly by the application. Otherwise, an incorrect or erroneous operation may be performed on the user's bank account.
- **Checking that all mandatory fields are filled in by the user:** For example, if email address is a mandatory field in a web form, it must not be left blank. If the field is left blank, a blank value may be saved in the database, thus affecting further processing based on this data.
- **Check the type of the data:** For example, if the Age field is an `integer` type, characters cannot be accepted as its value. This is because the processing that will be done using the `integer` age value cannot be done using characters. Therefore, the type of the data entered must match the type defined in the application.

Input validation is extremely important to ensure that the application and its data are secure and accessible at all times. Invalid input can lead to erroneous or incorrect operations being performed, corruption of data, loss of data, or even a system crash. Input validation is even more important in web applications because of their vulnerability to malicious users and code over the Internet. To understand the importance of accepting only valid input data in web applications, consider the following scenario.

An employee is using his company's online system to access salary information. To access the information, he needs to enter his employee ID and press the Submit button. If any unauthorized person wishes to access this information, he can possibly do so by entering some trick input deliberately in the employee ID field of the web page, such as 'EID7687' OR '1'='1'. This technique is known as SQL Injection. At a first glance this may seem like junk data that will not retrieve any valid data from the database. However, if you observe it in context of a SQL query that will be sent to the database when the user submits this data, the query will evaluate to `SELECT * from Employees WHERE ID= 'EID7687' OR '1'='1'`. So, in this case, all information from the Employees table may be retrieved because the clause `1=1` will always evaluate to `true` and in an OR operation, `false` OR `true` is always `true`.

However, the preceding situation can be prevented by fixing the length of string allowed in the employee ID field and probably defining its format, and then checking the length and format of the entered data against the fixed values.

As an application developer, you can only advise a user to enter data in the correct format. However, it is your responsibility to enforce input validation by writing code for it. Input validation code can be executed at the client side as well as the server side. Usually a client-side scripting language, such as JavaScript, is used by applications to perform client-side validations, and a server-side technology is used for server-side validations. However, client-side scripting is not completely reliable because the user can turn off the support for scripting language on the browser, or there may be situations where the browser itself does not support client-side scripting. In such cases, the application is exposed to vulnerability by the lack of input validation, as discussed before.

MORE INFORMATION

For more information on the process of input validation at client and server side, refer to MSDN.

The new validation framework in ASP.NET has added features and flexibility for both client- and server-side validation. However, before you learn about client- and server-side validation differences, let us discuss how ASP.NET provides support for input validation through its server-side validation framework.

UNDERSTANDING ASP.NET SERVER-SIDE INPUT VALIDATION SUPPORT

Input validation should be enforced in an application by writing code that checks the desired conditions on the input before processing it. This has been a tedious and time consuming task for application developers in classic ASP and many other programming platforms. However, ASP.NET greatly simplifies this process for developers by providing a flexible validation framework consisting of various validation controls.

Validation controls are controls that can be associated with any ASP.NET server control or HTML control (together referred to as input control further) on a page that perform a specific type of input validation. In case the validation check fails, the validation controls display an error message. An input control can be associated with one or more validation controls, and the input must satisfy all of the validation criteria related to the associated validation controls. Figure 7-1 shows the validation controls available in the Visual Studio 2008 Toolbox.

Figure 7-1

Validation controls in Visual Studio 2008 Toolbox



TAKE NOTE *

An input control can be validated if it has an associated validation property. The validation property is that property of the control, whose value needs to be validated. All controls that can be validated have the name of their validation property defined in the **ValidationProperty** attribute.

Unlike input controls, validation controls are normally invisible UI elements on a form. The visual elements associated with validation controls are the error messages displayed in case the validation check fails.

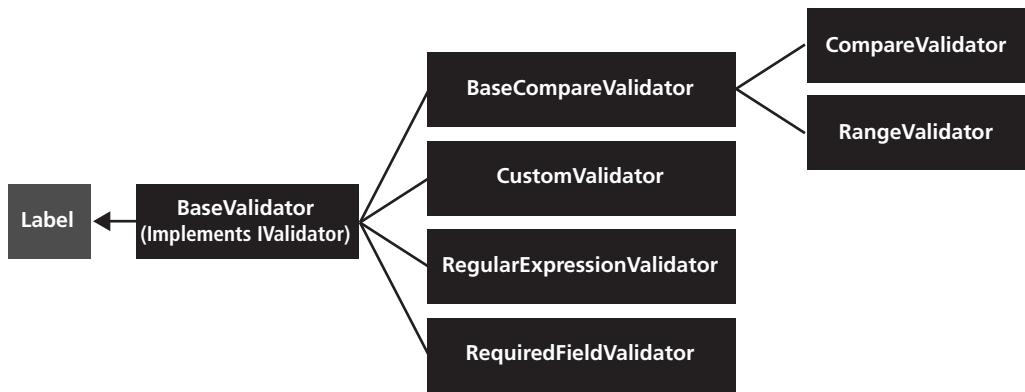
Validation controls are used for server- as well as client-side validation. The usage of the validation controls varies slightly in both cases. Let us understand the server-side validation object model.

USING THE SERVER-SIDE VALIDATION OBJECT MODEL

The object model is made up of the validator classes representing various validation controls. These classes belong to the `System.Web.UI.WebControls` namespace. The abstract base class from which all other validator classes inherit is `BaseValidator`. Figure 7-2 shows some of the main classes in the server-side validation object model of ASP.NET.

Figure 7-2

Classes in server-side validation



USING THE BASEVALIDATOR AND BASECOMPAREVALIDATOR CLASSES

The **BaseValidator** class is an abstract base class that is derived from the **Label** class. It implements the **IValidator** interface. This interface provides the properties and methods that must be implemented by any object that performs input validation. Table 7-1 describes some of the commonly used properties of the **BaseValidator** class.

Table 7-1

BaseValidator class properties

VALIDATOR	DESCRIPTION
ControlToValidate	Set the value of this property to the ID of the input control that needs to be validated. The specified input control and the validator instance must be on the same page, user control, or template, as applicable. By default, the property is set to an empty string. In order to prevent the application from throwing an exception, you must set a valid value for this property, except in the case of a CustomValidator control, in which case, you can leave it blank.
IsValid	Set the value of this property manually if required. Usually, the value is set to true or false based on whether the validation check has succeeded or failed. The value of the property is set to true by default and is updated when the results of validation are obtained.
Display	<p>Define how space is allocated to display the error message in case the validation performed by the validator instance fails. The value of this property can be one the members of the System.Web.UI.WebControls.ValidatorDisplay enumeration. These are the values defined in this enumeration:</p> <p>None: Indicates that the error message will not be displayed in line with the input control and will instead be displayed at a common location by using a ValidationSummary control, discussed later in the lesson.</p> <p>Static: Indicates that appropriate space will be allocated beforehand for the display of error messages for the validator instance (the default value for the Display property). Setting the Display property to this value prevents any changes in the layout of the page due to error messages displayed on validation failure. This value works in the case of client-side validation on Internet Explorer 4.0 and later. In all other cases, it works the same way as the dynamic value.</p> <p>Dynamic: Indicates that space for error messages on validation failure needs to be allocated dynamically. Setting the Display property to this value causes change in the layout of the page due to error messages displayed on validation failure.</p>
Text	Set the description of the error message displayed on validation failure in the validation control by setting the value of the overloaded Text property. Because the BaseValidator class inherits from the Label class, the other overloaded form of the Text property defines the contents of a Label control.

(continued)

Table 7-1 (continued)

VALIDATOR	DESCRIPTION
ErrorMessage	Set the value of this property, instead of the Text property, to the description of the error message if you want to display the error message associated with the validation control in a common location (in the ValidationSummary control). If you do not set the Text property and only set the ErrorMessage property, the value of the ErrorMessage property is displayed (in-line) in the validation control in addition to being displayed in the ValidationSummary control.
SetFocusOnError	Set the value of this property to true if you want the input focus to automatically shift to the associated input control when validation fails. By default, the value of this property is set to false . In case more than one validation fails on a page, and the value of this property is set to true for more than one of these validation controls, the focus will be set to the input control associated with the first validation control.
RenderUplevel	Retrieve the value of this read-only property to determine whether the user's browser supports Uplevel rendering. An Uplevel browser is one that supports Microsoft Internet Explorer Document Object Model (DOM) version 4 or later and ECMAScript version 1.2 or later. Client-side validation using the ASP.NET validation object model is possible only when Uplevel browser rendering is enabled.
ValidationGroup	Set the value of this property to the name of a validation group with which the validation control will be associated. A validation group is a set of input controls whose ValidationGroup property is set to the same name. These input controls must have the ability to postback or submit data to the server and cause validation after postback when their CausesValidation property is set to true . Examples of such controls are Button, DropDownList, ListBox, TextBox, and BulletedList. The default value of the property is an empty string indicating that its value is not set.
EnableClientScript	Set the value this property to true or false to indicate whether you want to enable or disable client-side validation for the validation control represented by the validator instance. By default the value of this property is set to true . However, client-side validation is performed only when the browser supports it, even when the value of this property is set to true .

TAKE NOTE*

Text and **ErrorMessage** properties also accept HTML tags as their value. Therefore, you can use multimedia elements to display the validation error messages.

+ MORE INFORMATION

Like all other controls, the validation controls also have **Enabled** and **Visible** properties defined for them. However, the values of these properties have a significant effect on the validation. For understanding the difference and effect of these properties, refer to MSDN.

The most crucial method of the **BaseValidator** class is the **Validate** method. It performs the validation check on the associated input control. The value of the **IsValid** property is updated after this method is invoked to perform the validation check and obtain its results. Note that this method is invoked automatically with an input control that can postback to the server has its **CausesValidation** property set to **true**. The method is invoked after postback but before any event handlers are invoked. However, when you are working with validation controls programmatically, you also can invoke this method explicitly. For example, if you are setting the value of validation controls properties at runtime, you may need to invoke the **Validate** method explicitly after the values have been obtained.

TAKE NOTE*

The **BaseValidator** class also provides an **EvaluateIsValid()** method, which is used to check whether the value in the input control is valid. This method returns a Boolean value that must be overridden by the derived class if you plan to define your own custom validation class. For more information, refer to MSDN.

The `BaseCompareValidator` class is the abstract base class for validators that perform typed comparisons. Although the `BaseCompareValidator` class inherits the validation-related properties and methods from the `BaseValidator` class, the following properties and methods of the `BaseCompareValidator` class are specific for type comparisons:

MORE INFORMATION

For details about how this is used, refer to the `BaseCompareValidator` . . . `CutoffYear` Property section on MSDN.

X REF

For more information on cultures, refer to Lesson 8, “Working with Globalization, Localization, and Accessibility.”

- **Type:** This property defines the datatype to which values being compared by the validator are converted before they are compared. The value of this property is a member of the `System.Web.UI.WebControls.ValidationDataType` enumeration, which can be `String`, `Integer`, `Double`, `Date`, or `Currency`. The default value is `String`. If conversion to the specified datatype is not possible before comparison, the validation will fail.
- **CutOffYear:** This read-only property retrieves the maximum year that can be represented using the last two digits of the year. The value of this property is based on the value of the `Calendar.TwoDigitYearMax` property.
- **CanConvert()**: This method checks whether the string, passed as the first argument, can be converted to the datatype passed as the second argument, considering the current culture defined. The culture affects the format of the data. The second argument is a value from the `System.Web.UI.WebControls.ValidationDataType` enumeration. The method returns `true` if the conversion is possible and otherwise returns `false`. The method has another overloaded form that accepts a third argument, which is a Boolean value, that when set to `true` defines the possibility that conversion is checked by considering an invariant culture.
- **Convert()**: This method converts the string passed as the first argument to an object of the datatype passed as the second argument. The second argument is a value from the `System.Web.UI.WebControls.ValidationDataType`. An instance of the `Object` class is passed as the third argument to store the converted object. The conversion is sensitive to the data formats defined by the current culture settings. To make the conversion insensitive to culture settings, you can use the overloaded form of the method that accepts a Boolean value (set to `true` for invariant culture) as the third argument and the `Object` instance as the fourth argument.
- **Compare()**: This method compares the strings passed as the first two arguments and returns a Boolean value based on the results of the comparison. The comparison operation to be used to compare the two strings is passed as the third argument. The comparison operation is a value from the `System.Web.UI.WebControls.ValidationCompareOperator` enumeration, which can be `Equal`, `NotEqual`, `GreaterThan`, `GreaterThanOrEqualTo`, `LessThan`, `LessThanOrEqualTo`, and `DataTypeCheck`. If the last value of the enumeration is used, the second argument passed to the `Compare()` method is ignored, and the datatype check is performed on the first argument. The method also accepts a fourth argument that defines the datatype of the values being compared. The comparison performed is sensitive to the culture setting. If you want to ignore the culture setting, you can use the overloaded form of the `Compare()` method, `Compare(String, Boolean, ValidationCompareOperator, ValidationDataType)`. The Boolean arguments can be set to `true` to consider an invariant culture.
- **GetFullYear()**: This method accepts the two-digit integer representing the short form of a year as an argument, such as 82, 78, and 99, and returns the four-digit full form such as 1999. The value returned by the method is based on the value of the `CutOffYear` property.

All derived classes of the `BaseValidator` and `BaseCompareValidator` classes represent the validation controls in ASP.NET.

USING VALIDATION CONTROLS

You can use the validation controls in your application. The validation controls are `RequiredFieldValidator`, `RangeValidator`, `CompareValidator`, `RegularExpressionValidator`, and `CustomValidator`.

USING THE REQUIREDFIELDVALIDATOR CONTROL

This control is used to implement mandatory fields in a page. The control ensures that the user enters or selects a value in the associated input control. If the validation fails, an error message is displayed based on the property settings for the validator. To add a RequiredFieldValidator control to your page, you can drag and drop it from the Visual Studio 2008 Toolbox. You can also declare it using the following declarative syntax:

```
<asp:textbox id="txtName" runat="server"/>
<asp:RequiredFieldValidator id="valMandatory" runat="server"
ControlToValidate="txtName" Text="* Please enter a name for your
account">*</asp:RequiredFieldValidator>
```

These lines of code declare a textbox and txtName. Next, a RequiredFieldValidator control named valMandatory is added, and its attributes are defined. The ControlToValidate attribute is set to txtName to associate the textbox with the validator. This will ensure that users are required to enter a value in the textbox. Finally, the error message description is assigned as the value of the Text attribute.

TAKE NOTE *

All validation controls except RequiredFieldValidator pass validation if a valid value is not entered in the associated input control.

You can also add the control programmatically by creating an instance of the **RequiredFieldValidator** class and setting its properties. The class inherits the methods and properties from the **BaseValidator** class. An important property of the **RequiredFieldValidator** class is **InitialValue**. It is used to define or retrieve the initial value set in the associated input control. The validation performed by RequiredFieldValidator fails if the user does not change the initial value defined for the input control. This means that when the associated input control loses focus or when the page is posted back to the server, if the value of the **InitialValue** property has not changed, the validation fails. The initial value is set as an empty string by default.

USING THE RANGEVALIDATOR CONTROL

This control is used to check whether the value input in the associated input control falls within a range. You can also add this control from the toolbox, declaratively or programmatically. The following properties of the **RangeValidator** class are crucial for validation:

- **MaximumValue:** Defines the upper limit of the range against which the input control will be validated.
- **MinimumValue:** Defines the lower limit of the range against which the input control will be validated.

In addition, the **RangeValidator** class inherits properties and methods used for comparisons from the **BaseCompareValidator** class.

The following is an example declaration of a RangeValidator control:

```
<asp:textbox id="txtAge" runat="server"/>
<asp:RangeValidator id="valAgeRange" runat="server"
ControlToValidate="txtAge" MaximumValue="50" MinimumValue="18"
Type="Integer" Text="* Please enter an age between 18 and 50.
">*</asp:RangeValidator>
```

This code example first declares a textbox named txtAge where the user can input an age value. Then, a RangeValidator control is declared, and its attributes are set. The ControlToValidate attribute is set to txtAge to associate the textbox with the validator. The maximum and minimum values of the range within which the entered age must fall are specified in the **MaximumValue** and **MinimumValue** attribute as 18 and 50. Next, the datatype used to compare the input with the maximum and minimum values of the range is defined as integer in the **Type** attribute. Finally, the error message description is provided as the value of the **Text** attribute.

USING THE COMPAREVALIDATOR CONTROL

This control is used to compare the user input to another value. The value can be a constant defined in code, or it can be the value of another input control. Moreover, you can also check

if the datatype of the input can be converted to the datatype specified in the Type property of the CompareValidator control. Note that the **CompareValidator** class is derived from the **BaseCompareValidator** class and inherits the Type property as explained already, among other comparison-related properties and methods. Some useful properties specific to the **CompareValidator** class are:

- **ControlToCompare**: Compares the value of the associated input control with the value of another input control. The ID of the latter input control is set as the value of this property.
- **ControlToValidate**: Represents the ID of the control that should be validated.
- **Operator**: Defines the type of comparison operation. The value of the property is a member of the **ValidationCompareOperator** enumeration. If the value of this property is set as **DataTypeCheck**, the values of the preceding two properties are ignored.
- **ValueToCompare**: Compares the value of the associated input control with a constant value. The datatype of the value of this property is a string that is converted to the appropriate datatype (specified in the Type property) for comparison. If the conversion of the datatype fails, an exception is thrown (except if the operator is **DataTypeCheck** because in that case the validation fails). You should not set both **ControlToCompare** and **ValueToCompare** at the same time. If you do so, **ControlToCompare** is given precedence over **ValueToControl**.

The following code example demonstrates the usage of the **CompareValidator** control using declarative syntax. Like other validation controls, you can also add this control from the toolbox or work with it programmatically:

```
<asp:textbox id="txtID" runat="server"/>
<asp:CompareValidator id="valLoginCmpr" runat=vserver"
ControlToValidate="txtID" ValueToCompare="user1" Type="String"
Text="* The entered ID does not match the ID entered at the time
of registration."*></asp:RangeValidator>
```

In this code example, a textbox named txtID is defined to allow the user to enter an ID. Then, a **CompareValidator** control is defined and associated with the textbox by setting the **ControlToValidate** attribute to txtID. The value of the textbox has to be compared to a constant value, user1. This is done by setting the value of the **ValueToCompare** attribute to user1. Note that the **Operator** attribute has not been explicitly set because the default value of Equal needs to be used. Finally, the datatype to be used for the comparison is set as String in the **Type** attribute.

USING THE REGULAREXPRESSIONVALIDATOR CONTROL

This control is used to match an input to a specific pattern of characters. The pattern of characters is specified by the use of a regular expression. You can add a **RegularExpressionValidator** from the toolbox using declarative syntax. Alternatively, you can add a control programmatically by using the **RegularExpressionValidator** class that is derived from the **BaseValidator** class. Apart from the inherited properties, an important property of the **RegularExpressionValidator** class is **ValidationExpression**. This property defines the regular expression against which the associated input has to be validated, as a String datatype.

The following code example shows how to add a **RegularExpressionValidator** control to validate an email address:

```
<asp:textbox id="txtEmail" runat="server"/>
<asp:RegularExpressionValidator id="regExEMail"
ControlToValidate="txtEmail" ValidationExpression="\w+([-.\'])\w+@\w+([-.\'])\w+([-.\'])\w+"
Text="Enter a valid e-mail address."
EnableClientScript="False" runat="server"/>
```

In this code example, a textbox named txtEmail is defined first, followed by a **RegularExpressionValidator** control. The validator is associated with the textbox by setting the **ControlToValidate** attribute. Thereafter, the **ValidationExpression** attribute is set to a regular expression.

For example, an email address is valid when defined as `username@domainname.domainextension`. Therefore, to check the validity of the entered email address, you can match it to a regular expression `\w+([-.\'])\w+\@(\w+([-.\'])\w+)*\.\w+([-.\'])\w+)*`.

Note that the client-side validation in this example is disabled. You need to do this in order to avoid validation of the email address on the client side. This is because the regular expression matching at the client side may be different from that at the server side. You will learn more about client-side validation in the later sections.

USING THE CUSTOMVALIDATOR CONTROL

This control is used to perform custom validation checks. The validation check logic is written by the programmer. This is useful if you want to perform a validation check beyond what can be done by other validator controls or if you want to use information generated at runtime to perform validation checks. For example, if you want to validate an input against a list of values in a text file or database. An advantage of this control is that you do not have to necessarily associate it with an input control; therefore, it can be used for validating other controls such as check boxes. You can add this control from the toolbox, declaratively, or programmatically by using the `CustomValidator` class. The `CustomValidator` class is derived from the `BaseValidator` class. Apart from the inherited members, three noticeable members of this class are:

- **ValidateEmptyText:** This property should be set to `true` if you want to perform the custom validation check even when the input control has an empty string value. For other validation controls, except `RequiredFieldValidator`, when the value of the associated input control is an empty string, the validation check is skipped and the validation is considered passed. However, for the `CustomValidator` you can control this by using the `ValidateEmptyText` property. In that case, the validation check will actually be performed using the empty string value, and validation results will be returned as passed or failed depending on the custom logic.
- **ServerValidate:** This event is raised when the validation check is performed on the server side. To provide custom validation check, you need to add a handler for this event. The validation will be performed when the page is posted back in response to a button `Click` event or any event that causes a postback.
- **OnServerValidate():** This method raises the `ServerValidate` event. It accepts the name of the method written to perform custom validation as an argument.



HANDLE THE CUSTOMVALIDATOR CONTROL

Handling the `CustomValidator` control is a little more complex than other controls. To do so:

1. Add the `CustomValidator` control.
2. Write a method for handling the custom validation. The method must be declared as:
`<MethodName (Object src, ServerValidateEventArgs serverArgs)>`

If the validator is associated with a specific input control, the `Value` property of the `ServerValidateEventArgs` structure will allow you to access the value to be validated as `serverArgs.Value`. In case the validator is associated with multiple controls, you can use the properties of the controls' classes to retrieve the input values. Note that the first argument of the method refers to an HTML tag, such as `` in which the validator control is contained on the page.

3. Set the value of the `IsValid` property of the `ServerValidateEventArgs` structure as `serverArgs.IsValid = true` (or false, as required) in the custom method based on the results of the validation.
4. Invoke the `OnServerValidate` method (for programmatic) or set the `OnServerValidate` attribute (for declarative) passing the name of the method that contains the code for custom validation.

Another control in the toolbox, under the Validation section, is `ValidationSummary`. This control is not used to perform validations but it is used with validation controls

to display the summary of all error messages at a common place. When you set the `ErrorMessage` property of a validation control, the specified value is displayed in the `ValidationSummary` control. Before exploring its usage, let us understand the members of the `ValidationSummary` class.

The `System.Web.UI.WebControls.ValidationSummary` class has the following commonly used properties:

- **DisplayMode:** Defines the format in which the error summary is displayed in the `ValidationSummary` control. The value can be a member of the `System.Web.UI.WebControls.ValidationSummaryDisplayMode` enumeration. The enumeration has three members, `List`, `BulletList`, and `SingleParagraph`.
- **ShowMessageBox:** Displays the error summary in a message box, in addition to being displayed on the page. This property value is only applicable when client-side scripting is enabled. So, the summary is displayed in a message box when the `ShowMessageBox` property and the `EnableClientScript` property are set to `true`.
- **ShowSummary:** Defines whether the error summary is displayed on the web page. By default, this is set to `true`. You can set both, the `ShowMessageBox` property and the `ShowSummary` property at the same time.
- **ValidationGroup:** Defines the set of validation controls for which the `ValidationSummary` control displays error summary. You can set a name for the validation group as the value of this property. Then, `ValidationSummary` control will display messages for all the validation controls whose `ValidationGroup` property is set to the same value. If the value of this property is not defined, the `ValidationSummary` control will display messages for all validation controls on the page that do not belong to any validation group.

You can combine the validation group feature with cross-page postbacks. You can select the validation group, which should be the basis for validating a page. For example, say you have a search box in your header that is present on every page. When you press “search,” you have a validator that makes sure that the query text box is not empty. Now let’s say you are on a page with a form such as a registration page. The registration controls also have validators, but you don’t need to check whether the search box is empty when the user clicks on “create new account.”

In addition to adding validation controls and setting their display-related properties, you need to determine when to check the `IsValid` property and start processing the validated data. For this, you need to understand the validation framework supported by the `page` object.

UNDERSTANDING THE PAGE OBJECT VALIDATION SUPPORT

In order to get server-side input validation working, you also need to use the validation related members of the `Page` object. These members are:

- **IsValid:** This property defines whether all validation controls on the page successfully passed the validation check. If set to `true`, you can proceed with processing the validated input data. If even one validation control’s `IsValid` property is set to `false`, the value of the `Page` object’s `IsValid` property will also be `false`. You should check the value of this property in the event handler of the control that caused a postback.
- **Validators:** This property retrieves a collection of all objects that implement the `IValidator` interface, which includes all the validation controls on the page. The datatype of the property is `System.Web.UI.ValidatorCollection`.
- **Validate():** This method is invoked on submission of the input data to the server (such as on button click or postback), before any event handler is invoked.

To understand how the server-side validation object model and the `Page` object work together to implement input validation on the server side, consider an example. Consider that you have created a Bank Account Registration Form with the fields, Name, Address, Age, Email, and Existing Account number. There is a Submit button.

You now have to implement input validation for this form as follows:

- The name, address, age, email address must be mandatory fields.
- The age must be between 21 and 45.
- The email address must be in a valid format.
- The existing account number, if entered, must match an entry in the text file storing a list of all existing account numbers.
- You want all error messages to be displayed in a ValidationSummary control.



IMPLEMENT SERVER-SIDE INPUT VALIDATION

To implement input validation on the form:

1. Identify the validator controls applicable for each input and declare them, as shown in the table. Table 7-2 lists the fields and the validator controls that should be used on them.

Table 7-2

Fields and the corresponding Validator controls

FIELD	VALIDATOR CONTROL	CODE
Name and Address textbox <code><asp:TextBox ID="txtName" runat="server" Width="173px"></asp:TextBox></code> Note: Similar code must be added for the Address textbox.	Two RequiredFieldValidator controls should be defined for the Name and Address textboxes to force user to enter valid values.	<pre><asp:RequiredFieldValidator ID="rname" runat="server" ControlToValidate="txtName" Display="Dynamic" ErrorMessage="Please Enter Name" SetFocusOnError="true">*</asp:RequiredFieldValidator></pre> <p>Note: Similar code must be added for the Address textbox.</p>
Age textbox <code><asp:TextBox ID="txtAge" runat="server" Width="173px"></asp:TextBox></code>	A RequiredFieldValidator control is used to force the user to enter a valid value, and a RangeValidator ensures that the value entered is between 21 and 45.	<pre><asp:RequiredFieldValidator ID="vage" runat="server" ControlToValidate="txtAge" ErrorMessage="Please Enter Age" SetFocusOnError="true">*</asp:RequiredFieldValidator></pre> <pre><asp:RangeValidator ID="rvage" runat="server" ControlToValidate="txtAge" ErrorMessage="Please Enter Age between 21 and 45" MaximumValue="45" MinimumValue="21" SetFocusOnError="true" Type="Integer">*</asp:RangeValidator></pre>
Email address <code><asp:TextBox ID="txtEmail" runat="server" Width="173px"></asp:TextBox></code>	A RequiredFieldValidator control is used to force the user to enter a valid value, and a RegularExpression Validator checks the format of the address.	<pre><asp:RequiredFieldValidator ID="vemail" runat="server" ControlToValidate="txtEmail" ErrorMessage="Please Enter Email" SetFocusOnError="true">*</asp:RequiredFieldValidator></pre> <pre><asp:RegularExpressionValidator ID="rev" runat="server" ControlToValidate="txtEmail" ErrorMessage="Please Enter a Valid Email Address" ValidationExpression="^([a-zA-Z0-9_\.\-\+]+@[a-zA-Z0-9\.\-\+]+\.[a-zA-Z]{2,4})\$">*</asp:RegularExpressionValidator></pre>

Table 7-2

FIELD	VALIDATOR CONTROL	CODE
		<pre>ControlToValidate="txtEmail" ErrorMessage="Invalid Email Address" SetFocusOnError="true" ValidationExpression="\w+([-.\'])\w+([-.\'])*\.\w+([-.\'])*\>*" </asp:RegularExpressionValidator></pre>
Account Number textbox <code><asp:TextBox ID="txtAccno" runat="server" Width="173px"></asp: TextBox></code>	A CustomValidator control must be used to check the entered value against the values in a file.	<pre><asp:CustomValidator ID="CustomValidator1" runat="server" ControlToValidate="txtAccno" ErrorMessage="Account Does Not Exist in Database" onservervalidate="CustomValidator1_ServerValidate">*</asp: CustomValidator></pre>

2. Next, to complete the design of the page, add a ValidationSummary control, as shown in the code:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"  
HeaderText="Page has the following errors:" BorderStyle="None"  
Font-Bold="true" Font-Italic="true" />
```

3. Next, in the code-behind file, you need to implement the custom validator method CustomValidator1_ServerValidate() as shown:

```
protected void CustomValidator1_ServerValidate(object source,  
ServerValidateEventArgs args)  
{  
    if (txtAccno.Text.Trim() != string.Empty)  
    {  
        string fName = Server.MapPath("Validator_text.txt");  
        // Path to text file  
        StreamReader testTxt = new StreamReader(fName);  
        string allRead = testTxt.ReadToEnd();  
        // Reads the whole text file to the end  
        testTxt.Close();  
        // Closes the text file after it is fully read.  
        string regMatch = txtAccno.Text.Trim();  
        // String to search for inside of text file. It is case  
        sensitive.  
        if (allRead.Contains(regMatch))  
            // If the match is found in allRead  
            {  
                args.IsValid = true;  
            }  
        else  
            {  
                args.IsValid = false;  
            }  
    }  
}
```

In this code example, a file named Validator_text.txt is read, and the input value of the txtAccno control is matched against the entries in the file. If the input value is found in the file, the IsValid property is set to **true**, or otherwise **false**.

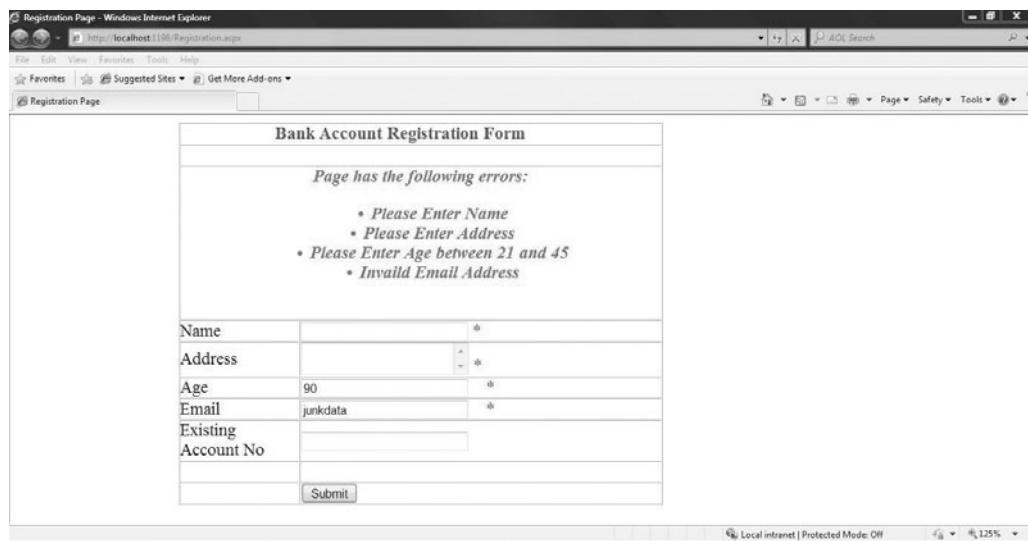
- Finally, you should check the **IsValid** property of the page in the **Click** event handler of the **Submit** button, as shown in the following code example:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        // Accept and process data
    }
}
```

Figure 7-3 shows a sample output of the application in case of validation failure.

Figure 7-3

Sample output



Understanding Client-Side Validation Support

As discussed already, ASP.NET validation framework also supports client-side validation. Client-side validation means that the input validation will be performed on the client. This technology prevents submission of a page that contains invalid input to the server, thereby improving the speed of the feedback on user's input. However, an important feature of the validation framework in ASP.NET is that the server-side validation is performed regardless of whether the client-side validation is performed. This is done for security reasons where client-side scripting is not enabled on a client or if the application is susceptible to security attacks.

Client-side validation is implemented in a client-side scripting language such as JScript or VBScript. Note that VBScript is only supported by Internet Explorer. It can be performed if the **EnableClientScript** property of a validation control is set to **true** and the browser is uplevel. The validation controls have built-in support for client-side validation.

If client-side validation needs to be performed, a reference to a script library (named WebUIValidation.js) containing validation code for all the validation controls is sent to the browser in the page. The script library is downloaded on the client, and the code in it is used to perform client-side validation. As a result, you may not have to make too many changes in your application to support client-side validation, if you have already implemented server-side input validation.

However, there are some crucial differences in the validation object model and specific validation controls that you must know. In the client-side validation model, there are four main global variables that help you control the validation logic:

- **Page_IsValid:** This Boolean variable corresponds to the `IsValid` property of the page as in the server-side model.
- **Page_Validators:** This array corresponds to the `Validators` property of the page as in the server-side model.
- **Page_ValidationActive:** This Boolean variable is used to programmatically turn client-side validation on or off. When set to `true`, client-side validation is performed (provided it is enabled and supported by the browser).
- **IsValid:** This is a property of each validator on the client side, which indicates whether the validation check was successful or not.

CERTIFICATION READY?

Implement client-side validation and server-side validation.

2.4

Table 7-3

Client-side Validation controls

VALIDATION CONTROL	DESCRIPTION
BaseCompareValidator and its derived controls, RangeValidator and CompareValidator	If the <code>Type</code> property is set to the <code>Date</code> value, client-side validation is performed only if the currently set Calendar type is Gregorian . Otherwise, server-side validation is performed.
RegularExpressionValidator	The regular expression checking criteria on the client might vary slightly from the checking done with the help of the <code>System.Text.RegularExpressions.Regex</code> class on the server. However, the JScript regular-expression syntax, when used for regular expression validation, is more likely to generate the same results as in the case of server-side validation through the <code>Regex</code> class.
CustomValidator	Supports client-side validation features. To implement custom validation on the client side, in addition to the server-side custom method, you must write a client-side custom method in JScript or VBScript. The client-side custom method must accept two arguments. The first argument refers to the HTML element <code></code> in which the control is rendered. The second argument refers to the object whose <code>IsValid</code> and <code>Value</code> properties aid in the validation logic. The name of the client-side custom method is set as the value of the <code>ClientValidationFunction</code> property.

TAKE NOTE*

The client-side script for validators supports only those input controls that are visible. Any controls that are hidden, or are inside other grouping controls and are hidden, can be validated only through server-side validation.

■ Performing Site Management

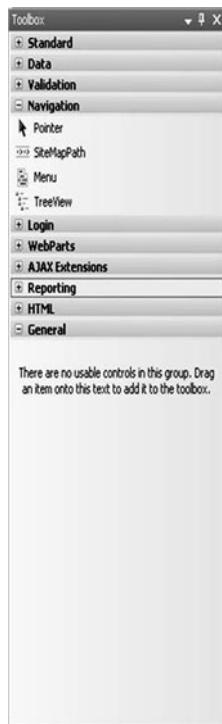


THE BOTTOM LINE

Web sites are often hierarchical in nature, and pages are sometimes nested several layers deep. Navigating through such pages is sometimes problematic. Therefore, when designing Web sites, it is important to give the user ways to easily find out where they are and where else can they navigate to. To address this issue, some Web sites include a menu bar across the top of the page with links to all the areas on the site. Some sites provide a tree structure to help navigation. A few others include a “bread-crumb” trail showing where you are currently and how to get back to where you started from. ASP.NET supports all these navigation methods using navigation controls, site map data source, and site map provider architecture.

Figure 7-4

Controls for navigation



ASP.NET controls and classes simplify the implementation and management of such navigation features in a Web site. The main characteristics of **site navigation** features supported by ASP.NET are discussed next:

- **Simplified implementation of site navigation:** Site navigation can be implemented with the help of three main navigation controls: SiteMapPath, Menu, and TreeView. Because of these controls, the programmer's job has been simplified to a great extent. Programmers can add these controls by dragging and dropping them from the toolbox, either declaratively or programmatically, as with other controls in ASP.NET. Figure 7-4 shows the controls under the Navigation section in the Visual Studio 2008 Toolbox.
- **Better management of site navigation features:** Site navigation is usually simple to implement on most platforms when the size of the Web site is small. However, as the size of a Web site grows, and the level of nesting web pages becomes deeper, it becomes tedious for programmers to keep track of them. This leads to site navigation problems such as broken links and unintuitive navigation, further leading to a less-than satisfactory user experience. To overcome this problem, ASP.NET allows you to store links to all pages and site structure information in a centralized location through the **SiteMap** class and its related classes. Thus, site structure-related information and links can then be read by all navigation controls from a centralized location and edited or removed from here only. Moreover, rules can be defined to display or hide a link to a page in a navigation control. This centralized mechanism facilitates better site management.
- **Consistent navigation:** With a centralized approach and support for master pages, programmers can now develop a consistent navigation user interface. Consistent navigation means that information related to structures and links will always be kept in central location and it will not vary across pages. Moreover, if navigation controls need to be repeated across pages of the Web site, they can be designed on the master page two that they will appear consistent on all pages.

To implement navigation in a web application, it is suggested that you:

1. Create one or more site maps, which are logical structures defining the hierarchy of pages in a Web site. You must keep the site map updated for all new pages added or pages deleted from the Web site.
2. Add navigation controls that read structural information about the Web site from the site map.

In this lesson, you will learn how the preceding steps are implemented in ASP.NET in detail.

Working with Site Maps

In ASP.NET a site map is a map of the Web site that stores the navigational paths between the pages of the Web site. Therefore, the first step in implementing a consistent site navigation is to add a site map. By default, a site map can be created as an XML file named web.sitemap and stored in the root directory of the web application. You can create other custom site map files that use underlying storage mechanisms other than XML. You will learn about custom site maps later in the topic. First, let us discuss the structure of the web.sitemap file. The web.sitemap file structure is shown in Figure 7-5.

Figure 7-5

Web.sitemap file structure



As shown in Figure 7-5, a web.sitemap file should have the site map root element that encloses `siteMapNode` elements. Each `siteMapNode` element represents a page in the Web site. A `siteMapNode` element can further have any number of child `siteMapNode` elements. The nesting of these elements defines the hierarchy relation between pages. A valid web.sitemap file must have only one `siteMapNode` element under the `siteMap` element that encloses all the other `siteMapNode` elements in the file.

For each `siteMapNode` element, you can define attributes that state the name of the web page that it represents, the URL of the page, and the description.

To understand how you can do this, consider the Web site of an online catalog for an apparel retailer. The Web site has two main pages that you can navigate to from the home page: Catalog and Contact Us. From the Catalog page, you can navigate to the Men's Clothes, Women's Clothes, and Children's Clothes pages. From the Contact Us page, you can navigate to the Locate Retail Store and Buy a Franchise pages.



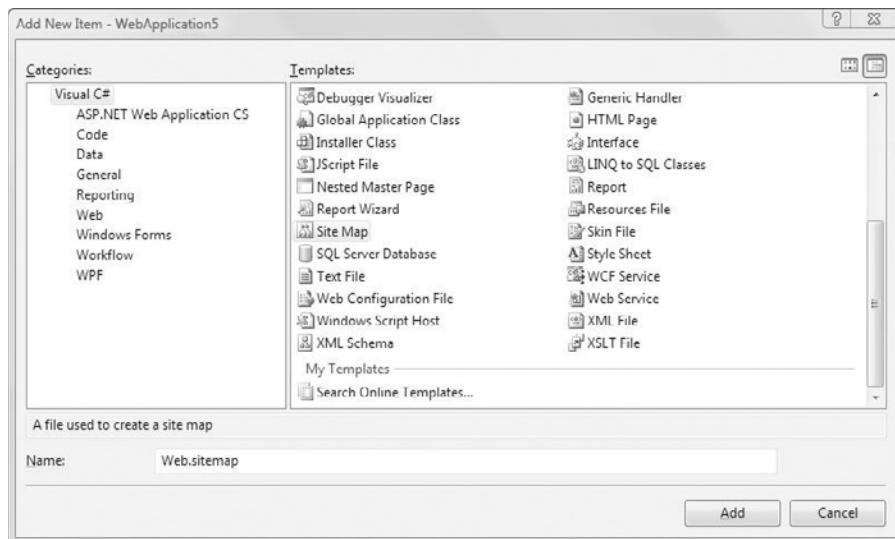
CREATE A WEB.SITEMAP FILE

To represent the map of this Web site in a web.sitemap file, you will perform the following steps:

1. In the Solution Explorer, right click the web application root, and select Add > New Item from the shortcut menu. The Add New Item dialog box appears.
2. In the Add New Item dialog box, under Templates, select Site Map as shown in Figure 7-6.

Figure 7-6

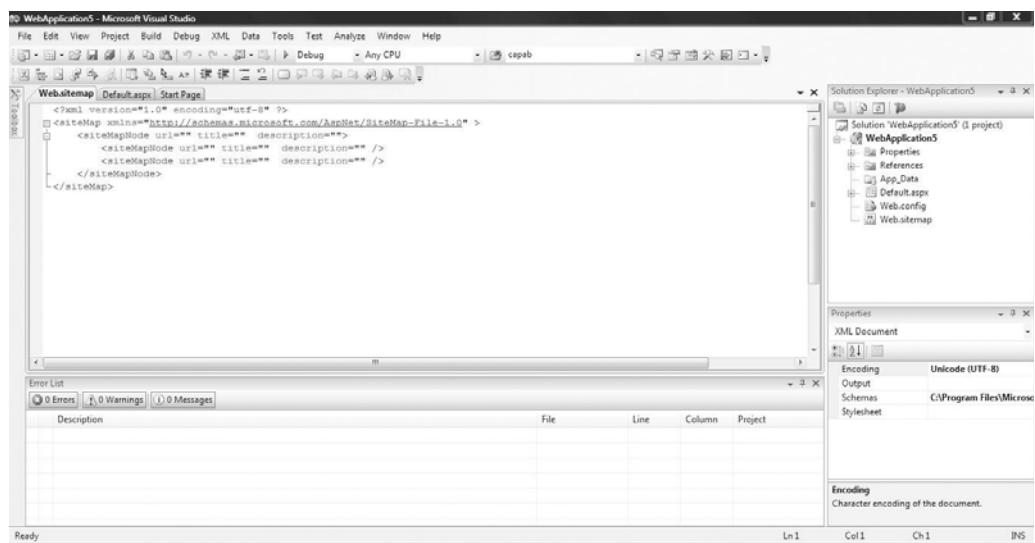
Add New Item dialog box



3. Click the Add button. The web.sitemap file will be added to the application as shown in Figure 7-7.

Figure 7-7

Adding sitemap



4. Add the following code to the web.sitemap file:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-
    File-1.0" >
    <siteMapNode url="~/default.aspx" title="Home"
        description="Home page">
```

```
<siteMapNode url "~/catalog.aspx" title="Product Catalog"
    description="Main catalog page">
<siteMapNode url "~/mens.aspx" title="Mens Catalog"
    description="Mens apparel catalog" />
<siteMapNode url "~/womens.aspx" title=" Womens Catalog "
    description="Womens apparel catalog" />
<siteMapNode url "~/children.aspx" title="Childrens Catalog"
    description="Childrens apparel catalog" />
</siteMapNode>
<siteMapNode url "~/contactus.aspx" title="Contact Us"
    description="Contact information page">
<siteMapNode url "~/locate.aspx" title="Locate A Retailer"
    description="Locate retailer through a map" />
<siteMapNode url "~/franchisee.aspx" title=" Franchisee Information"
    description="Get information for a franchisee" />
</siteMapNode>
</siteMapNode>
</siteMap>
```

In the preceding code, note the value of the URL attribute. The ‘~/’ indicates that the page is in the root directory. The URL attribute can be left blank. However, for a valid sitemap file, it should not be duplicated. In addition, to the URL, other attributes defined for each `siteMapNode` element are `title` and `description`. You can set the value of the `title` attribute to the text that you want to display as a hyperlink for the page. So, in “bread crumbs” or a menu on the Web site, the link to the `catalog.aspx` page will be displayed as Product Catalog. The value of the `description` attribute is used as a tooltip in navigation controls that support displaying a tooltip, such as `SiteMapPath` (used for adding “bread crumbs”). In addition, it also serves as reference documentation for developers.

MORE INFORMATION

For detailed information about how paths can be specified for pages of Web sites, refer to the ASP.NET Web Site Paths section on MSDN.

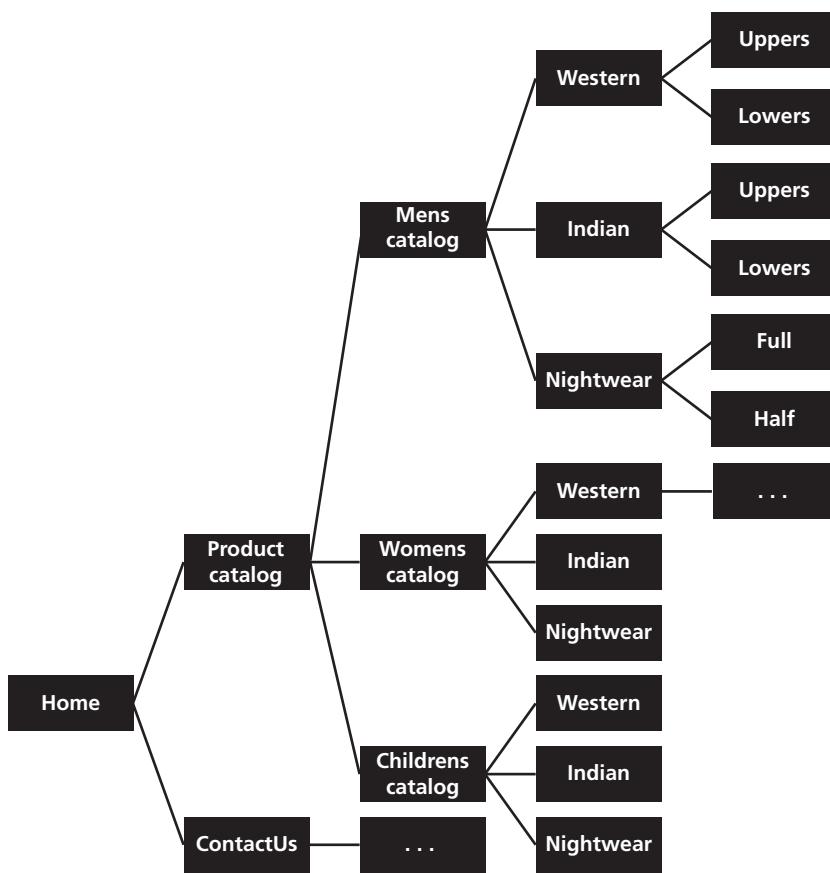
Note that a site map provider is used to manage the storage of site map information and communicate it to navigation controls. The default site map provider is `XmlSiteMapProvider`. In addition to creating the sitemap file, the configuration file, `web.config` must also have the site map provider settings configured. The following code shows the settings for the default provider:

```
<siteMap defaultProvider="XmlSiteMapProvider" enabled="true">
    <providers>
        <add name="XmlSiteMapProvider"
            description="Default XML Sitemap Provider"
            type="System.Web.XmlSiteMapProvider"
            siteMapFile="Web.sitemap"
            />
    </providers>
</siteMap>
```

You are not restricted to use one sitemap file in an application. You can use multiple sitemap files in the same application. These sitemap files may reside in the root folder or any subfolder of the application. In order to use multiple sitemap files, you can set a `siteMapFile` attribute for a `siteMapNode` element. The `siteMapFile` attribute allows you to refer to another sitemap file inside a sitemap file. When you define the value of this attribute, you do not need to define the `url`, `description`, or `title` attributes. The node, at which the second sitemap file name is specified, then uses the second sitemap file. For example, consider the apparel retailer’s Web site again. Suppose, you want to add nested sets of pages for, Western, India, and Nightwear, under Men’s, Women’s, and Children’s apparel catalog pages. Further, you want to nest and display each apparel type as shown in Figure 7-8.

Figure 7-8

Multiple site maps



In this case, you may want to put the nodes for Western, Indian, Nightwear, and so on in a separate child sitemap file. To do so, you can edit the code for `web.sitemap`, defined previously, using the `siteMapFile` attribute as follows:

```

<siteMapNode url="~/mens.aspx" title="Mens Catalog" description="Mens apparel catalog" >
    <siteMapNode siteMapFile="~/MenApp.sitemap"/>
<siteMapNode/>

<siteMapNode url="~/womens.aspx" title=" Womens Catalog " description="Womens apparel catalog" >
    <siteMapNode siteMapFile="~/WomenApp.sitemap"/>
<siteMapNode/>

<siteMapNode url="~/children.aspx" title="Childrens Catalog" description="Childrens apparel catalog" >
    <siteMapNode siteMapFile="~/ChldrnApp.sitemap"/>
<siteMapNode/>
  
```

TAKE NOTE *

Optionally, you can configure multiple site maps in the `web.config` file also. For details, refer to MSDN.

In this code, separate sitemap files are defined for the hierarchy beyond Mens/Womens/ Childrens Catalog pages.

In addition to configuring multiple site maps that use the default XML provider for storing the site map, you can also create custom site map providers and use them. You may want to change to a custom provider for many reasons, such as when you want to store the site map in some other place such as a database or a simple text file. The custom site map provider functionality is implemented with the help of the `SiteMapProvider` class. When your site map has to use a customer provider, you can specify the name of the provider by setting the value of the `provider` attribute of the `siteMapNode` element. The custom provider set here must also be defined in the `web.config` file of the application.

USING THE SITEMAP CLASS

You can retrieve information stored in the sitemap file programmatically by using the members of the `System.Web.SiteMap` class. A `SiteMap` object (along with the `System.Web.SiteMapNode` class object) represents the entire sitemap file in memory. The `SiteMap` class provides the following useful static properties:

- **RootNode:** Retrieves the root node, which represents the top-level page of the Web site.
- **CurrentNode:** Retrieves the `siteMapNode` element of the sitemap file that corresponds to the currently requested page. The value type retrieved by this property is an instance of the `SiteMapNode` class. The class represents a node (web page) in the sitemap file. The `SiteMapNode` class also provides various useful properties and methods of programmatic manipulation. In addition to `Description`, `Title`, and `Url`, which correspond to the attributes of the `siteMapNode` element explained previously, for some other useful properties of the `SiteMapNode` class refer to MSDN.
- **Provider:** Identifies the default site map provider for the site map represented by the current `SiteMap` instance. The list of all providers listed in the configuration file can be obtained by using the `Providers` property. The `Providers` property retrieves `System.Web...:SiteMapProviderCollection` object.

The `Sitemap` class does not provide any methods. It offers an event for notifying the current node access called the `SiteMapResolve` event. Whenever any control retrieves the current node, the `SiteMap_SiteMapResolve` method executes if it is attached to the event handler. If there is no node that corresponds to a page, the method creates a new node and returns it. You can handle the `SiteMapResolve` event to modify the node returned when the `CurrentNode` property is accessed in code.

Now that you have learned the basics of creating site maps, we will discuss navigation controls and how site maps are used for navigation.

Working with Navigation Controls

You can implement site navigation with the help of three controls: `SiteMapPath`, `Menu`, and `TreeView`.

USING THE SITEMAPPATH CONTROL

This control is used to display a “bread-crumb” trail for a web page. A bread-crumb trail consists of links to the pages traversed by the user in the Web site’s hierarchy up to the page currently being viewed. It allows a user to directly traverse back to any page included in the chain by clicking on the link. The control is only used for navigational purpose and works with a site map. The main advantages of this control are that it uses minimal space on the UI and provides an intuitive navigation feature particularly for Web sites that have a deep hierarchy. However, it cannot be used for traversing forward from the current page.

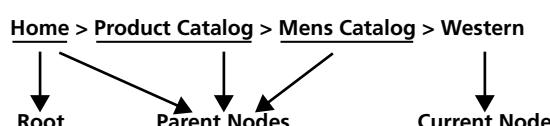
A `SiteMapPath` control consists of nodes that represent web pages. Each node is represented as a `HyperLink` or `Literal` control in the UI and can be of any one of the following types:

- **Current node:** Refers to the current web page that a user is viewing.
- **Root note:** Refers to the first page in the bread-crumb trail from which the user started navigating the current hierarchy of pages displayed in the trail.
- **Parent node:** Refers to any node that has further nested nodes or child nodes.

Figure 7-9 shows a conceptual bread-crumb trail with these three types of nodes.

Figure 7-9

Types of nodes



You can add a SiteMapPath control to your application by dragging and dropping from Visual Studio 2008 Toolbox, by using declarative syntax, or programmatically by using the `SiteMapPath` class. The commonly used properties of the `SiteMapPath` class are:

- **Provider:** Set the value of this property to an instance of the `SiteMapProvider` class, which represents the default site map provider. If you do not explicitly set a value, the default `XmlSiteMapProvider` class is used.
- **SiteMapProvider:** Set the value of this property to an instance of the `SiteMapProvider` class, which represents the site map provider to be used for the `SiteMapPath` control. If the value of this property is not defined, the value of `Provider` property is used.
- **PathSeparator:** Set this property to specify the string used to separate nodes in a bread-crumb trail rendered on a page for the current `SiteMapPath` control. The default value of this property is ">", which means that nodes in the bread-crumb trail will be delimited by the > character as `Root Node > Parent Node 1 > Parent Node 2 > Current Node`. You can also retrieve the value of the `PathSeparatorStyle` property of the `SiteMapPath` class to identify the style used for the separator. The style can be set by using declarative syntax as follows:

```
<asp:SiteMapPath id="spmCatalog" Runat="Server" PathSeparator=" ->
    PathSeparatorStyle-Font-Size="10pt"
    PathSeparatorStyle-ForeColor="#167AB0" />
```

In addition, you can also define a control template to be used for the separator by setting the `PathSeparatorTemplate` property of the `SiteMapPath` class. However, in that case the `PathSeparator` and `PathSeparatorStyle` properties are not applicable:

- **PathDirection:** You can display the nodes in a bread-crumb trail from root to current or from current to root nodes. The default direction is from root (leftmost in the bread-crumb trail) to current node (rightmost in the trail). You can however specify the desired direction by setting the value of the `PathDirection` property. The value of this property can be any member of the `System.Web.UI.WebControls.PathDirection` enumeration. The enumeration has two members `RootToCurrent` and `CurrentToRoot`. You should set the `PathSeparator` accordingly if you change the direction.
- **CurrentNodeStyle:** You can use this property to retrieve the style used to display the title of the current node in the bread-crumb trail. The style for the current node can be set using declarative syntax as in the following example:

```
<asp:SiteMapPath id="spmCatalog" Runat="Server" PathSeparator=" ->
    CurrentNodeStyle-ForeColor="#167AB0" CurrentNodeStyle-Font-
    Italic="true" CurrentNodeStyle-Font-Underline="False"/>
```

Instead of style, you can set a control template for the current node by setting the `CurrentNodeTemplate` property. If this property is set, the title text to be displayed for the current node and the style defined in the `CurrentNodeStyle` property are ignored. Similarly, you can also retrieve/set the style and template for the root node by using the `RootNodeStyle` and `RootNodeTemplate` properties. In addition, you can also retrieve/set the style and template for all nodes by using the `NodeStyle` and `NodeTemplate` properties. When the style and template are defined for all nodes as well as explicitly for the root or current node, then the styles are merged into a single style by first applying the style for all nodes then overriding that style for the root/current node specifically by the style defined for the root/current node. When templates are defined, they cause all style settings to be ignored for the particular node(s):

- **ParentLevelsDisplayed:** Controls the number of parent nodes displayed for every current node in a bread-crumb trail by setting the value of the `ParentLevelsDisplayed` property to an `integer` value. By default, the value is set to `-1`, which means that no limit is defined. However, in many Web sites, you may want to limit this to avoid wasting space and to prevent confusion caused by large numbers of nodes in a bread-crumb trails.

- **RenderCurrentNodeAsLink:** Controls whether the current node is displayed as a hyperlink by setting the value of the `RenderCurrentNodeAsLink` property to `true` or `false`. By default, this property is set to `false`, and the current node is not displayed as a hyperlink.

A commonly used method of the `SiteMapPath` class is `InitializeItem()`. This method is used to initialize a node based on its type, in the bread-crumb trail represented by the `SiteMapPath` control. The node is represented by an instance of the `SiteMapNodeItem` class, which is a Web server control wrapper for the `SiteMapNode` class. The method ‘initializes’ means that it identifies as the right type of control that should be used to render the node and applies any styles and templates defined for it. To better understand this, let us explore how the method functions. The method accepts a `SiteMapNodeItem` instance as an argument and initializes it as shown in Table 7-4.

CERTIFICATION READY?
Configure providers.
1.1

Table 7-4Using the `InitializeItem` method

NODE TYPE	CONTROL TYPE CREATED TO RENDER IT	ADDITIONAL METHOD ACTION FOR INITIALIZATION
Current	If the <code>RenderCurrentNodeAsLink</code> property is set to <code>true</code> , a <code>HyperLink</code> control is created to represent the current node, otherwise a <code>Literal</code> control is used. Note: All controls created for the types of nodes given in this table are created as a child control of the <code>SiteMapPath</code> control and can be accessed by iterating over the value of the <code>Controls</code> property.	The style/template is applied, as described previously, if the <code>CurrentNodeStyle</code> / <code>CurrentNodeTemplate</code> / <code>NodeStyle</code> properties are set.
Root	A <code>HyperLink</code> control is created.	The style/template is applied, as described previously, if the <code>RootNodeStyle</code> / <code>RootNodeTemplate</code> / <code>NodeStyle</code> properties are set.
Parent	A <code>HyperLink</code> control is created.	The style/template is applied, as described previously, if the <code>NodeStyle</code> / <code>NodeTemplate</code> properties are set.
PathSeparator	A <code>Literal</code> control is created to represent the path separator.	The style/template is applied, as described previously, if the <code>PathSeparatorStyle</code> / <code>PathSeparatorTemplate</code> properties are set.

MORE INFORMATION

In addition to the properties and methods, a useful event of the `SiteMapPath` class is the `ItemCreated` event. The `ItemCreated` event is raised by the `OnItemCreated(SiteMapNodeEventArgs args)` method when a `SiteMapNodeItem` instance is created. You can define an event handler for this method if you want to customize the created item before it is associated with its data source. To see an example usage of this event along with the `InitializeItem()` method, refer to the `SiteMapPath...::OnItemCreated` Method section on MSDN.

USING THE MENU CONTROL

This is a general control that is also used for navigational purposes in addition to posting back data to the server for executing a command, such as Save. As a navigation control, it allows you to display links to web pages in top-level and nested menus. It can work with the

site map with the help of the `SiteMapDataSource` class. Its main advantage is that it shows you the entire or part of the hierarchy of the Web site and allows you to select and directly navigate to any web page in the hierarchy. In terms of space usage on the UI, it takes space on an as-needed basis—menus pop out when users point the mouse pointer at a nested-menu item. To conserve space, existing elements are overlapped (horizontally and vertically) on the web page as the menus pop out. It is useful for Web sites that have a shallow-medium hierarchy because too many nesting levels in the menu are difficult to remember considering that they only become visible when you point to their parent menu item.

The `Menu` control represents a menu on a web page. Menus are either static or dynamic. A static menu is always displayed whether users select an item or not, such as the main menu in Visual Studio 2008. A dynamic menu is displayed only when a user selects a menu item, and it may disappear after a specified time if the focus is removed from it. The `Menu` control can be used to display both types of menus. Before you learn how the `Menu` control is used for site navigation, let us explore it in detail.

A menu is made up of root, parent, and child menu items. A menu created in ASP.NET has the following structure:

- Level 0 or root menu items at the top level
- Menu items at the subsequent levels. A menu item at a level x can be a child of a parent menu item at the level $x-1$.

You can add a `Menu` control by dragging and dropping it from the Toolbox, declaratively by using the `<asp: Menu>` element, or programmatically by using the `System.Web.UI.WebControls.Menu` class. Menu items are represented by the `MenuItem` class.

In Visual Studio 2008, you can easily create menus in the design view as shown in Figure 7-10.

Figure 7-10

Creating menus

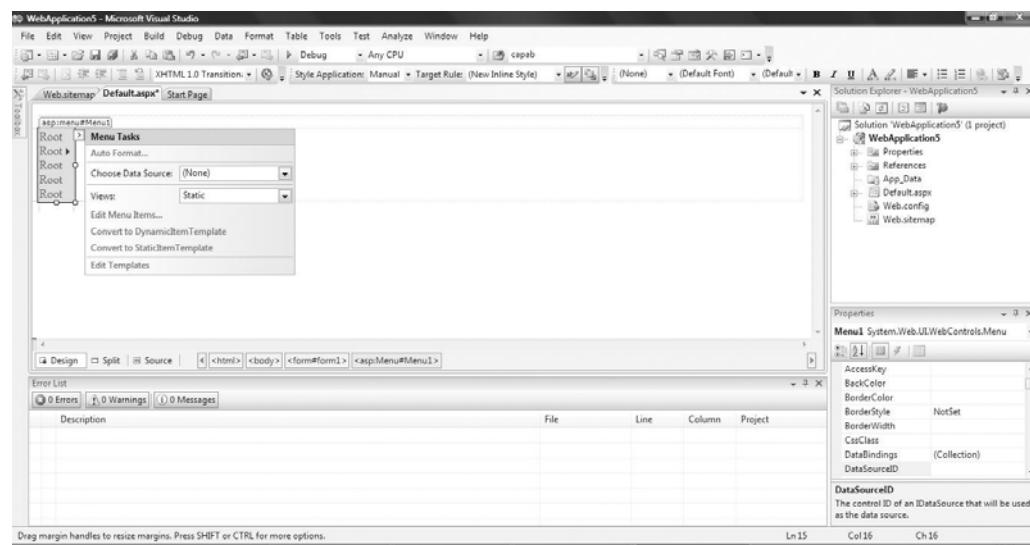


Figure 7-10 shows a menu added by dragging and dropping from the Navigation section of the Toolbox. You can set the properties for the menu and menu items easily in the design view as shown in the preceding figure.

The following code example demonstrates how you can declaratively create a menu in ASP.NET:

```
<asp:menu id="RetailMenu" runat="server">
<items>
    <asp:menuitem navigateurl="Home.aspx"
        text="Home" />
    <asp:menuitem navigateurl="Catalog.aspx"
        text="Product Catalog" />
    <asp:menuitem navigateurl="Contactus.aspx"
        text="Contact Us" />
</asp:menuitem>
</items>
</asp:menu>
```

In the preceding code, the same top-level hierarchy as explained for the apparel retail scenario has been used to create a menu. Note how the `<asp:menuitem>` elements are nested in the main `<asp:menuitem>` element, which is further enclosed in the `<items>` element.

Programmatically, you can create similar menus by using the `Menu` and `MenuItem` classes.

Next, let us discuss the `Menu` and `MenuItem` classes in detail.

The `Menu` class has the following commonly used properties:

- **SelectedItem:** Retrieves the value of this property to identify the menu item currently selected by the user. The item is returned as an instance of the `MenuItem` class. You can use this property when the user selects an item from the menu and the `MenuItemClick` event is raised. The event handler of the `MenuItemClick` event can check the value of the property and perform the necessary operation. You can use the members of the `MenuItem` class to further process the selected menu item.
- **SelectedValue:** Retrieves the string displayed for the currently selected menu item in the menu. This value of this property is retrieved from the `Text` property of `MenuItem` instance representing the selected menu item.
- **Target:** Determines whether the results of a menu item click have to be displayed in a new window or in the window that currently has the focus. This property is used when menus are used for site navigation and the items represent hyperlinks to pages. The property can use any of the predefined string values, such as "blank" for a new window without frames and "self" for the frame that currently has focus. For a detailed list of values refer to MSDN.
- **DisappearAfter:** Makes the dynamic menus disappear automatically after a specific period of time if the user stops pointing the mouse pointer at them. The value of the property is specified in milliseconds. If the value of the property is set to -1, the menu will not disappear until the user explicitly clicks somewhere else. Note that even if the value of the property is set to a specific time period, the menu will disappear if the user clicks anywhere else.
- **StaticDisplayLevels:** Defines the menu levels up to which the menus will be displayed as static. By default the value of static levels is set as 1. For example, Figure 7-11 shows the difference between the values 2 and 3 for this property.

Figure 7-11

Difference in values of the `StaticDisplayLevels` property



All levels nested further than the specified value are displayed as dynamic menus.

- **MaximumDynamicDisplayLevels:** Defines the highest level at which dynamic menus are displayed. For example, if you set this value to 2 then two dynamic menus beyond the static menus will be displayed. If menu levels are defined even beyond these two dynamic levels, then the further additional levels are ignored and not displayed. By default, this property is set to 0.
- **Orientation:** Defines whether the menu is displayed horizontally or vertically. The value of this property can be any member of the

`System.Web.UI.WebControls.Orientation` enumeration, which includes `Orientation.Horizontal` and `Orientation.Vertical`. By default, the latter value is used. In addition to this property, there are various other properties that help you customize the appearance of a menu, such as `StaticMenuItemStyle`, `StaticMenuStyle`, `StaticHoverStyle` and others, as well as a similar set of properties for dynamic menus and for levels of menus.

The `Menu` class has the following commonly used methods:

- **`FindItem()`**: Retrieves the `MenuItem` instance representing the menu item at a specific location in the menu. The location is passed as a string containing the path from the root to the menu item.
- **`DataBind()`**: Associates a data source with the `Menu` control. The control can then read the menu data from the data source. The method has two overloaded forms `DataBind()` and `DataBind(Boolean toRaiseEvent)`. Also, there are other methods related to data binding, such as `SetItemDataBound()`, `SetItemDataItem()`, and `SetItemDataPath()`.

In addition, the `Menu` class also inherits from the `HierarchicalDataBoundControl` class. Therefore the data-binding related members, such as `DataSource` property and `GetDataSource()` method are also members of the `Menu` class. Use them when you want to associate a `Menu` control with a site map. You will learn about associating a menu with a site map shortly in the lesson.

Static menu items can be created by using the declarative syntax discussed before. However, if you want to create menu items programmatically, you need to use the `MenuItem` class. Each item in the menu can be programmatically represented by an instance of the `MenuItem` class. To instantiate the `MenuItem` class, you can use any of its overloaded constructors:

- **`MenuItem()`**: The default constructor.
- **`MenuItem(String menuText)`**: Accepts the initial value of the `Text` property of the `MenuItem` class.
- **`MenuItem(String menuText, String menuValue)`**: Accepts the initial value of the `Text` and `Value` properties of the `MenuItem` class.
- **`MenuItem(String menuText, String menuValue, String menuImageUrl)`**: Accepts the initial value of the `Text`, `Value`, and `ImageUrl` properties of the `MenuItem` class.
- **`MenuItem(String menuText, String menuValue, String menuImageUrl, String menuNavUrl)`**: Accepts the initial value of the `Text`, `Value`, `ImageUrl`, and `NavigationUrl` properties of the `MenuItem` class.
- **`MenuItem(String menuText, String menuValue, String menuImageUrl, String menuNavUrl, String menuItemTarget)`**: Accepts the initial value of the `Text`, `Value`, `ImageUrl`, `NavigationUrl`, and `Target` properties of the `MenuItem` class.

The commonly used properties of this class are:

- **`Depth`**: Identifies the nesting level at which the menu item exists in the `Menu` control.
- **`Parent`**: Retrieves the `MenuItem` instance representing the parent menu item of the current menu item.
- **`Text`**: Defines the string value displayed for the menu item in the `Menu` control.
- **`Value`**: Specifies any additional information related to the menu item. The string value of this property is not displayed on the `Menu` control but can be used for internal processing. If this property is not explicitly set, it is automatically set to the value of the `Text` property. Similarly, if the value of the `Text` property is not explicitly set, it is set to the value of the `Value` property if defined. Note that in order to allow the code to distinguish among menu items, the value of this property should be unique for each menu item.
- **`Selected`**: Retrieves the value of this `Boolean` property to determine whether the current menu item is selected.

- **Selectable:** Defines whether a menu item can be clicked or selected. This property is different from the Enabled property. If this property is set `false`, no processing occurs when the menu item is clicked. However, the child menu of a nonselectable menu item can be selectable and is still displayed. However, in the case of Enabled property, the menu item is displayed as disabled in the menu and its child submenu is not shown. Note that this property is applicable only when a menu item posts back on being clicked but not when the NavigationUrl property is set and the menu item is used for navigation on a web page.
- **NavigationUrl:** Uses a string value to represent a URL to which you want the user to navigate when clicking on a menu item.
- **ImageUrl:** Displays an image next to the textual name of the menu item in the Menu control by setting the value of this property to a URL for the image.
- **DataSourceID:** Associates the menu with a data source, such as site map.
- **ChildItems:** Retrieves menu items nested in the submenu of the current menu item. The items in submenu are called the child menu items, and the current menu item is the parent menu item. The value of the property is an instance of the `MenuItemCollection` class. You can use the methods of the `MenuItemCollection` class to add, delete, and further manipulate the child menu items. Note that the `MenuItem` instances comprising the main level 0 (root) menu can be accessed from the `Items` value (collection) property of the Menu control.
- **Target:** Defines the target type of window for a specific menu item. The Target property for the Menu control is applicable for all menu items.

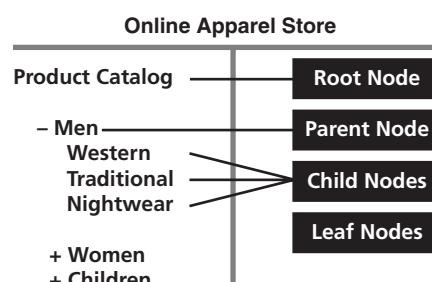
USING THE TREEVIEW CONTROL

This is a general control that is used for navigational purposes and to display data in a hierarchical format. As a navigational control, it allows you to view the hierarchy of the entire or part of the Web site in a tree view and click on any node to visit the corresponding page directly. Unlike a menu, it uses/releases vertical space in the UI as its nodes are expanded or collapsed. It is useful for Web sites that have a deep hierarchy.

The TreeView control is used to display data in a hierarchical structure. Each element of the control is called a node and can be of four types: root, parent, child, or leaf as shown in Figure 7-12.

Figure 7-12

Hierarchical node structure



A tree can have multiple root nodes. Each node can have associated text, an expansion/collapse symbol, and optionally a checkbox and an image. Note that the nodes (Except leaf nodes) in a tree view can be expanded and collapsed by clicking on the + and – symbols (default) against the node. Expanding a node means viewing all nodes under it.

Like the Menu control, the TreeView control can also be added from the toolbox, declaratively or programmatically. The `TreeView` class instance represents a tree view in a web page. The nodes of the tree view are represented by instance of the `TreeNode` class.

Some of the commonly used properties of the `TreeView` class are:

- **CheckedNodes:** Displays a check box against nodes in a tree view. This allows you to select multiple nodes in a tree view on which to perform a processing operation. The nodes whose check box is selected can be retrieved from the value of the `CheckedNodes`

property. The value is defined an instance of the `TreeNodeCollection` class. You can set the value of the `ShowCheckboxes` property if you want check boxes to be displayed against some or all nodes. The value of the property is a bitwise combination of the members of the `TreeNodeType` enumeration. The members of the enumeration are `All`, `None`, `Leaf`, `Root`, and `Parent`.

- **ExpandDepth:** Defines the level to which the tree view is expanded when it is first displayed. The default value of the property is `-1` and therefore a tree view displays all nodes by default.
- **MaxDataBindDepth:** Binds a tree view to a data source, as in the case of a menu. However, you can also limit the tree levels that are bound to the data source by setting this property. The count is taken from the root (inclusive) onwards.
- **Nodes:** Retrieves the `TreeNode` instance representing the nodes in the tree from the value of this property.
- **SelectedNode:** Retrieves the selected node in the tree view from the value of this property. The node is retrieved as an instance of the `TreeNode` class.
- **SelectedValue:** Retrieves additional information defined in the `Value` property of the `TreeNode` class for the currently selected tree node.
- **ShowExpandCollapse:** Displays expansion/collapsing symbols against parent nodes if you set this property to `true`. By default + and - signs are used but these can be customized by setting the `ImageSet` and related properties.
- **Target:** Serves the same purpose as the Menu control.
- **NodeIndent:** Defines how far right indented the child nodes will be displayed from the expanded parent nodes in the tree view. The value is an integer indicating the space in pixels between the leftmost edges of the parent and child nodes.

MORE INFORMATION

In addition, there are various style, template, and other appearance-related properties defined for the control. For details on these, refer to MSDN.

Some of the commonly used methods of the `TreeView` class are:

- **CollapseAll()**: Collapses all nodes in the tree view when called. Similarly the `ExpandAll()` method expands all nodes in the tree view.
- **FindNode()**: Retrieves a `TreeNode` instance representing a node at the specified location. The location is passed as an argument in the form of a path (value path) from the root to the current node.

As in the case of Menu control, the `TreeView` class also inherits properties and methods from the `HierarchicalDataBoundControl` class.

Let us now discuss the `TreeNode` class. You define the nodes in a tree view statically by using the declarative syntax. However, to define them dynamically, you can use the `TreeNode` class. The class has six overloaded constructors among which the following five are for common use:

- **TreeNode()**: The default constructor.
- **TreeNode(String tvText)**: Accepts the initial value of the `Text` property of the `TreeNode` class.
- **TreeNode(String tvText, String tcVal)**: Accepts the initial value of the `Text` and `Value` properties of the `TreeNode` class.
- **TreeNode(String tvText, String tcVal, String tvImageUrl)**: Accepts the initial value of the `Text`, `Value`, and `ImageUrl` properties of the `TreeNode` class.
- **TreeNode (String tvText, String tvValue, String tvImageUrl, String tvNavUrl)**: Accepts the initial value of the `Text`, `Value`, `ImageUrl`, and `NavigationUrl` properties of the `TreeNode` class.
- **TreeNode (String tvText, String tvValue, String tvImageUrl, String tvNavUrl, String tvNodeTarget)**: Accepts the initial value of the `Text`, `Value`, `ImageUrl`, `NavigationUrl` and `Target` properties of the `TreeNode` class.

Some of the properties of the `TreeNode` class serve the same purpose as in the `MenuItem` class, such as `Text`, `Value`, `Depth`, `Target`, `ImageUrl`, and `NavigationUrl`. The `TreeNode` class has the following commonly used properties:

- **Checked**: Selects (`true`) or clears (`false`) the check box of the node. You can also set the `ShowCheckBox` property to `Boolean` value to indicate whether you want to show a check box against the node.
- **ChildNodes**: Retrieves all the child nodes of the current node as an instance of the `TreeNodeCollection` class. Note that only the first level nodes are returned.
- **Expanded**: Determines whether the current node is expanded (`true`) or collapsed (`false`).
- **PopulateOnDemand**: Adds the child nodes of the current node dynamically if the value of this property is set to `true`. If the browser supports it, this can also be done on the client side when a user expands the current node.
- **SelectAction**: Defines the events raised on the selection of a node when the value of this property is set to a member of the `TreeNodeSelectAction` enumeration.
- **Selected**: Selects a node in code explicitly when the value of this property is set to `true`.

The commonly used methods are:

- **Expand()**: Expands the current node.
- **Collapse()**: Collapses the current node.
- **ExpandAll()**: Expands the current node as well as its child nodes.
- **CollapseAll()**: Collapses the current node as well as its child nodes.
- **Select()**: Performs the same action as the `Selected` property.

CERTIFICATION READY?
Implement data-bound
controls.
2.1

Now that you have learned how menus and menu items are used in an application. Let us explore how they can be linked with a site map.

LINKING THE SITEMAP CLASS AND MENU AND TREEVIEW CONTROLS

You define static items/nodes in the Menu/TreeView controls for site navigation if you know the entire structure beforehand. However, as the size of the Web site grows and the hierarchy becomes deeper, the Menus and TreeView controls may become scattered and lead to inconsistent and erroneous navigation. Therefore, it is a good design practice to associate the site map with the controls as a data source so that all changes related to the site structure need to be made only at one place and be reflected in all navigation controls.

Unlike SiteMapPath controls, Menu and TreeView controls cannot directly read the sitemap file through the provider. They need to use the `SiteMapDataSource` class, which acts as a data source for the Web server controls that support data binding and interfaces with the site map providers so that the controls can be used for navigation.

The properties of `SiteMapDataSource` class that are useful in this process are:

- **Provider**: Retrieves the site map provider associated with the `SiteMapDataSource` instance by default from the value of this property. The default `XmlSiteMapProvider` is used if no other provider is explicitly specified.
- **SiteMapProvider**: Retrieves the site map provider to which the control is bound from the value of this property.
- **ShowStartingNode**: Set the value of this property to `true` if you want the starting node to be retrieved from the sitemap file and displayed in the control that uses the `SiteMapDataSource` instance as its data source.
- **StartFromCurrentNode**: Set the value of this property to `true` if you want the site map to be retrieved from the current node onwards. The current node then becomes the starting node to be referenced for retrieving site map data through the provider.

- **StartingNodeUrl:** Set the value of this property to a URL of a specific node. The specified node is then used as the starting node to be referenced for retrieving site map data through the provider. Note that either the `StartFromCurrentNode` property can be set to `true` or the `StartingNodeUrl` can be set to a value other than `Empty` (default).
- **StartingNodeOffset:** Set the value of this property to a negative or positive integer. The value indicates the offset from the starting node that is referenced to retrieve the data from the site map. The node obtained based on the offset is then identified as the starting node for the associated Web server control. The positive and negative values are crucial because they define whether the subtree will be retrieved up in the hierarchy or down in the hierarchy from the initial starting node.

To understand how the `SiteMapDataSource` control is used to use a site map for menus and treeview, consider the site map discussed for the apparel retailer. The following code declares a menu and then associates it to a site map in a file named `Menu.aspx`:

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

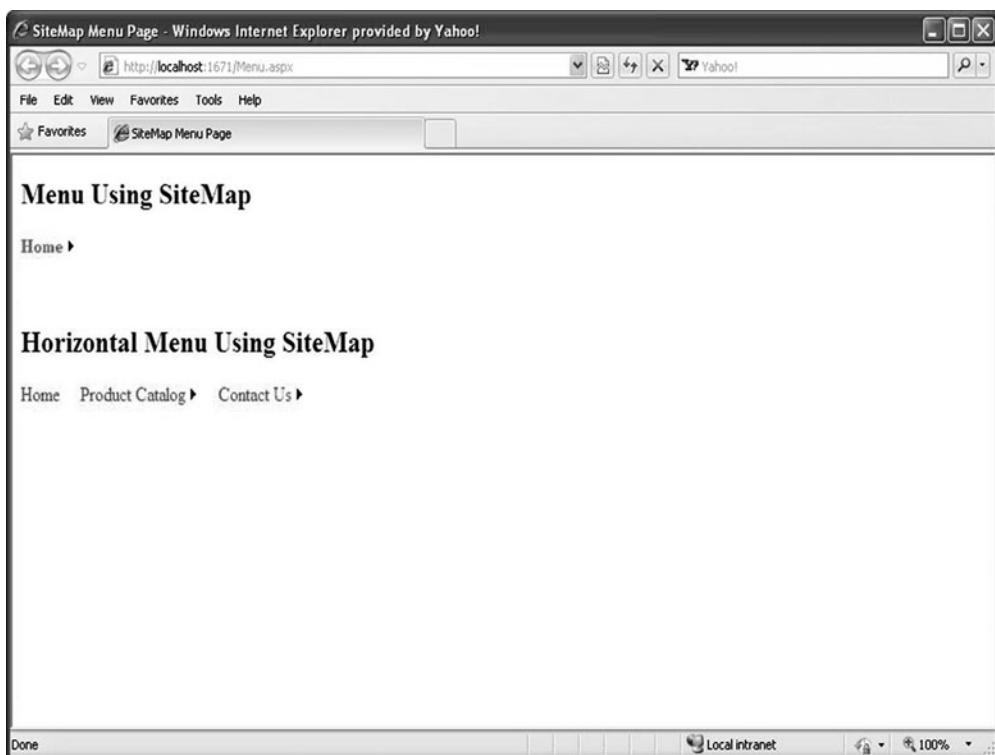
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>SiteMap Menu Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2> Menu Using SiteMap</h2>
            <asp:Menu ID="Menu1" Runat="server" Font-Bold=true
ForeColor="Magenta" DataSourceID="SiteMapDataSource1">
                </asp:Menu>
            <br />
            <br />
            <h2> Horizontal Menu Using SiteMap</h2>
            <asp:Menu ID="Menu2" Runat="server" ForeColor="Red"
DataSourceID="SiteMapDataSource1"
Orientation="Horizontal"
StaticDisplayLevels="2" >
                </asp:Menu>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
        </div>
    </form>
</body>
</html>
```

In the preceding code, two menus have been declared named `Menu1` and `Menu2`. Note how their appearance has been set by using the `ForeColor` and `Font-Bold` attributes. The orientation property of `Menu2` is specifically set to `Horizontal` instead of the default value of `Vertical`. The `StaticDisplayLevels` property has been set for `Menu2` to show the menu as static up to level 2. In addition, their `DataSourceID` property has been set to the ID of the `SiteMapDataSource` instance added to the code. Note that the `SiteMapProvider` property has not been set explicitly in the declaration of the `SiteMapDataSource` because the default site map provider is being used. The output of the preceding code is shown in Figure 7-13.

Figure 7-13

Output



Similarly, you can associate a `SiteMapDataSource` with a TreeView control. The following code associates a parent and three child maps (as discussed in the section on SiteMaps). The code also declares a `SiteMapPath` control that tracks the pages navigated by the user:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>TreeView Page Using SiteMap</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2></h2>
            <asp:SiteMapPath ID="SiteMapPath1" Runat="server"
PathSeparator=" ***">
                <CurrentNodeStyle Font-Bold="true" />
            </asp:SiteMapPath>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
            <h2>TreeView Using Multiple SiteMapPath</h2>
            <asp:TreeView ID="TreeView1" Runat="Server"
DataSourceID="SiteMapDataSource1" ParentNodeStyle-ForeColor="Chocolate"
LeafNodeStyle-ForeColor="Maroon" RootNodeStyle-ForeColor="Red"
ExpandDepth =0 >
                </asp:TreeView>
            </div>
        </form>
    </body>
</html>
```

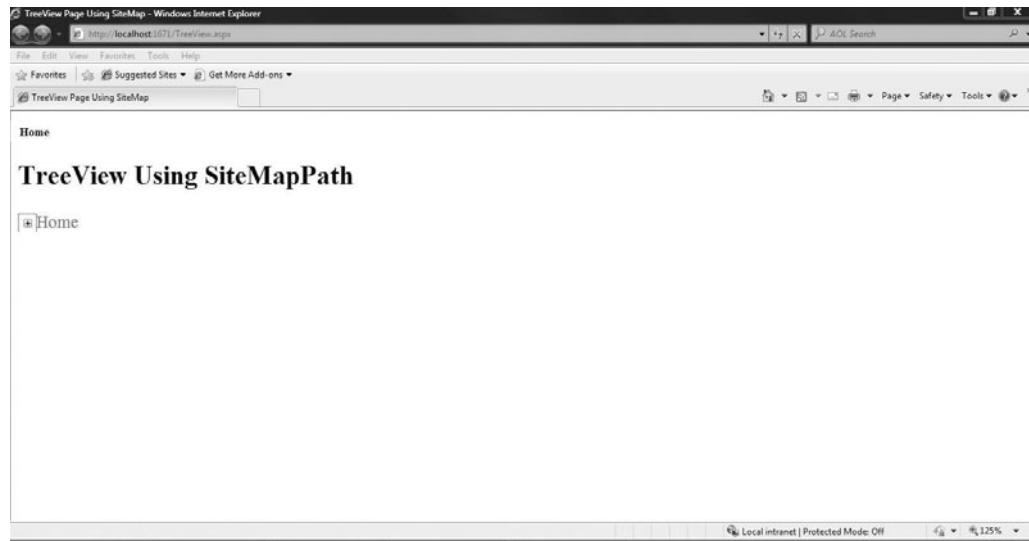
In the preceding code, note how the appearance-related and `ExpandDepth` attributes are set for the `TreeView` control. Setting the `ExpandDepth` to 0 will display all nodes collapsed. Also, again since the default site map provider is being used, the `SiteMapProvide` property is not explicitly set in the `SiteMapDataSource`.

In addition, a `SiteMapPath` control has been added to the page. The `PathSeparator` attribute has been set to `***` and the current node will be displayed as bold due to the `CurrentNodeStyle` attribute.

With the simple code given here, you can create a tree view, as shown in Figure 7-14.

Figure 7-14

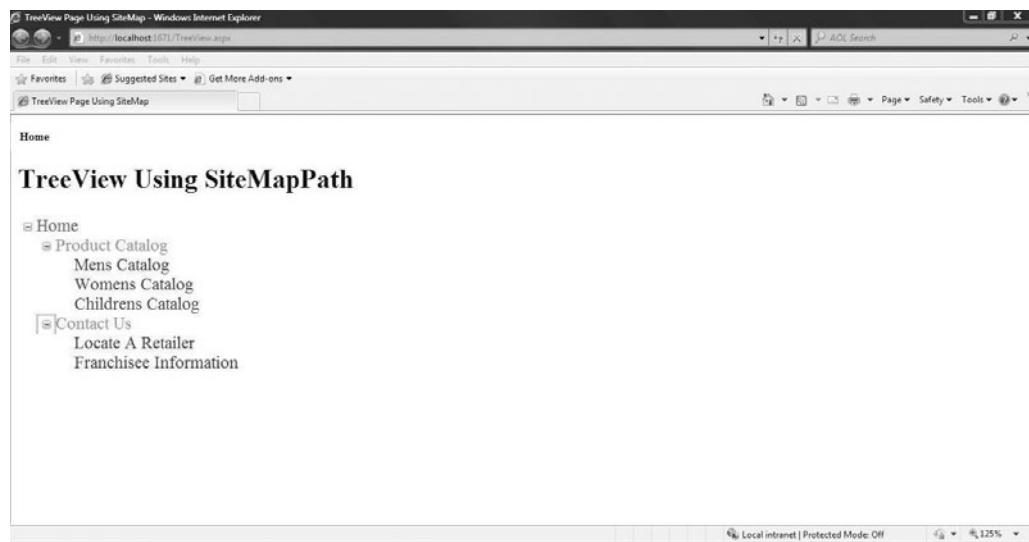
Tree view



When you expand the Home node and click the Product Catalog node, you can view the output as shown in Figure 7-15.

Figure 7-15

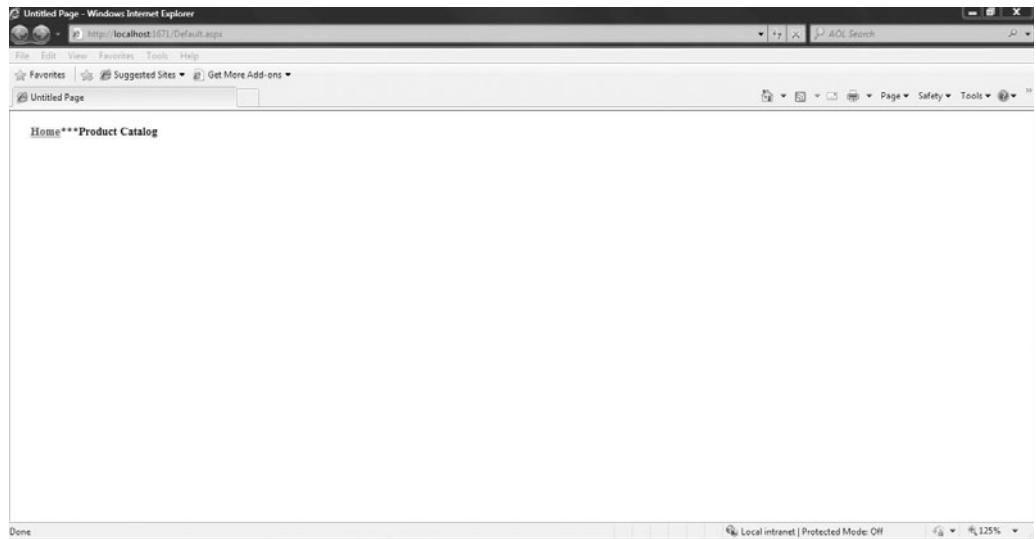
Expanding the tree view



Note that a SiteMapPath control needs to be added to all pages on which you want to display the bread-crumb trail. So, the default.aspx page that opens when you click on Product Catalog has a SiteMapPath control added to it with the same attribute set as defined for the Home page (Figure 7-16).

Figure 7-16

Product catalog page



Note that the current node, Product Catalog, is bold. The structure of the parent and child site maps is as follows:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="~/MTreeView.aspx" title="Home"
description="Home page">
        <siteMapNode url="~/default.aspx" title="Product Catalog"
description="Main catalog page">
            <siteMapNode url="~/mens.aspx" title="Mens Catalog"
description="Mens apparel catalog" >
                <siteMapNode siteMapFile="~/MenApp.sitemap"/>
            </siteMapNode>
            <siteMapNode url="~/womens.aspx" title=" Womens Catalog "
description="Womens apparel catalog">
                <siteMapNode siteMapFile="~/WomenApp.sitemap"/>
            </siteMapNode>
            <siteMapNode url="~/children.aspx" title="Childrens Catalog"
description="Childrens apparel catalog" >
                <siteMapNode siteMapFile="~/ChldrnApp.sitemap"/>
            </siteMapNode>
        </siteMapNode>
        <siteMapNode url="~/contactus.aspx" title="Contact Us"
description="Contact information page">
```

```

        <siteMapNode url "~/locate.aspx" title="Locate A Retailer"
description="Locate retailer through a map" />

        <siteMapNode url "~/franchisee.aspx" title=" Franchisee
Information" description="Get information for a franchisee" />

    </siteMapNode>
</siteMapNode>
</siteMap>

//Code for WomenApp.sitemap

<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="Women's Apparel" description="">
        <siteMapNode url="" title="Western" description="" />
        <siteMapNode url="" title="Indian" description="" />
        <siteMapNode url="" title="Nightwear" description="" />
    </siteMapNode>
</siteMap>

//Code for MenApp.sitemap

<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="Men's Apparel" description="">
        <siteMapNode url="" title="Western" description="" />
        <siteMapNode url="" title="Indian" description="" />
        <siteMapNode url="" title="Nightwear" description="" />
    </siteMapNode>
</siteMap>

//Code for ChildrenApp.sitemap

<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
    <siteMapNode url="" title="Children's Apparel" description="">
        <siteMapNode url="" title="Western" description="" />
        <siteMapNode url="" title="Indian" description="" />
        <siteMapNode url="" title="Nightwear" description="" />
    </siteMapNode>
</siteMap>

```

Note that for demonstration purposes, the URLs have been left blank in the site maps.

Understanding URL Mapping

URL mapping refers to the processing of associating a URL with another URL so that the client request for a URL can be redirected to the associated URL.

Consider that URL A is mapped to URL B:

- When a client request comes for a page referred to by URL A, the user is directed to the page referred to by URL B.
- When the client is redirected to URL B, the user sees URL A only in the address bar of the browser.

URL mapping provides the following benefits from a site navigation perspective:

- **Increases the user friendliness of the Web site:** URLs should ideally be easy to remember, short, and intuitive. Additionally, you should give web pages names that are easy for the developers to recognize and map them with more user-friendly URLs for the user. For example, you can call a page, AddItemToDataBase.aspx for the convenience of the developers; however, you can map it to another URL AddToCart.aspx for the user.

- **Increases the maintainability and manageability of the Web site:** As time passes, many Web sites grow. As more pages are added, deleted, modified, and moved, URLs may need to be mapped to maintain and manage the different pages. For example, suppose a page is being modified; while that page is being modified, the developers may consider mapping that page's URL to another URL that displays a message that the page is unavailable.
- **Increases the flexibility of the Web site:** You may need to rewrite URLs dynamically for client requests according to user preferences and redirect them to pages. Such flexibility can also be built in using the URL mapping functionality.

URL mapping, or URL rewriting, can be implemented at the server level or by using ASP.NET.

TAKE NOTE *

You can also use the `UrlRewrite` module from Microsoft that works for IIS7+ and gives you a lot more control and functionality when implementing URL mapping. To understand how URL mapping is done at the IIS level, refer to MSDN.

Let us discuss how URL mapping can be implemented in ASP.NET.

USING STATIC URL MAPPING IN ASP.NET

ASP.NET supports URL mapping in the web.config files. In order to map URLs, you need to use the following syntax in the web.config:

```
<system.web>
    <urlMappings enabled="true">
        <add url="new user-friendly URL"
            mappedUrl="Original page URL"/>
        <add.../>
    </urlMappings>
    ...
</system.web>
```

In the preceding code, the `<urlMappings>` elements includes `<add>` child elements where each child element defines a mapping. The first attribute of the `<add>` child element is `url`, which defines the URL that the users will enter in their browser. The second attribute, `mappedUrl` defines the original URL of the page that is mapped to the user-friendly URL. The following code shows an example of the preceding syntax:

```
<system.web>
    ...
    <urlMappings enabled="true">
        <add url="~/AddToCart.aspx"
            mappedUrl="~/AddItemToDataBase.aspx?ID=2 "/>
        <add.../>
    </urlMappings>
    ...
</system.web>
```

In the preceding code example, when the user requests the `AddToCart.aspx` page, the `~/AddItemToDataBase.aspx?ID=2` will be rendered. However, the URL will still show `AddToCart.aspx`.

You can also retrieve the user-friendly URL or the original URL to which it is mapped in your code if required. You can use the `RawUrl` property of the `Request` class to retrieve the user-friendly URL that the user requested in the browser. You can retrieve the original remapped URL by using the `Path` property of the `Request` class. You can also retrieve the

original remapped URL in an application-relative format by using the `AppRelativeCurrentExecutionFilePath` property of the `Request` class.

TAKE NOTE *

URL **remapping** and site map functionality can be used together to simplify navigation. The site map nodes contain the user-friendly URLs. Note that ASP.NET uses the `RawUrl` property to check whether a node has been defined in the site map. In case a node is not found, it is checked against the value of the `Path` property.

Note that this support is for static URL mapping only, where URLs have to be predefined without any support for wild cards or regular expressions. However, you can write custom URL rewriters in order to implement more flexible remapping.

+ MORE INFORMATION

You can also use HTTP Handler for URL mapping. For detailed information, refer to MSDN.

CERTIFICATION READY?

Implement session state, view state, control state, cookies, cache, or application state.

7.5

USING CUSTOM URL MAPPING IN ASP.NET

You can implement custom HTTP modules that handle URL mapping. Here are the basic steps to follow when implementing custom URL mapping using HTTP modules:

1. Create an `HttpModule` class that implements the `System.Web.IHttpModule` interface.
2. Add the new module to the `web.config` file.
3. Add configuration data related to mapping rules in `web.config`.
4. Add code in the implemented class to handle the incoming request and map it as per the rules defined in the configuration file.
5. Optionally, you may programmatically define rules for making the application more flexible.

TAKE NOTE *

The `VirtualPathProvider` class is another class that is useful for managing file paths. This class allows you to represent the web pages in a virtual hierarchy that differs from the actual physical hierarchy of the files in the file system. For information about the `VirtualPathProvider` class, refer to MSDN.

TAKE NOTE *

In your applications, you may need to provide a path to which you refer pages or files. You can provide absolute paths or relative paths to refer files. To convert absolute paths to relative paths, you can use the `VirtualPathUtility` class and manipulate paths in your application. For more information about using the `VirtualPathUtility` class, refer to MSDN.

■ Implementing State Management**THE BOTTOM LINE**

A crucial aspect when working with dynamic Web sites is **state management**. State management refers to maintaining a persistent state of data related to pages or applications across multiple trips to and from the server.

For example, suppose you are browsing through the shopping Web site of the apparel retailer. You visit the Men's Catalog, select an item, and add it to the shopping cart. Next, you navigate to the Women's Catalog and select an item and add it to the cart. This browsing requires trips between the client and server as pages are retrieved from the server based on the client request sent. In this case, to ensure that the item selected from the Men's Catalog page is remembered by the application when the user navigates to the Women's Catalog page, you'll need to implement state management.

Understanding States and State Management

Based on the access scope, you can configure state at three levels: application, client, and page. Application-level state refers to that persisted data that is globally accessible to all pages in the application. Application state can be shared across clients. Client-level state refers to

that persisted data that is specific to a client session and accessible only for that client. A client session includes all pages navigated by a user in a particular session with the server. Page-level state refers to the data that is persisted across postbacks of a page to itself.

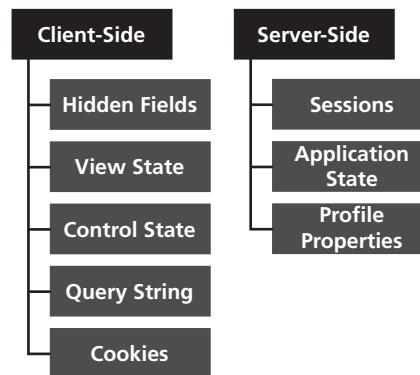
State management can be of the following types based on where the state is persisted:

- **Client-side state management:** Stores the state data on the client side. View state, control state, hidden fields, cookies, and query strings involve storing data on the client side in various ways.
- **Server-side state management:** Stores the state data on the client side. Server-side state management can be accomplished by using application state, session state, and profile.

Figure 7-17 shows the different management techniques that fall under each type of state management.

Figure 7-17

State management techniques



Note that all the preceding types of states are lost when an application is restarted. If you close all pages of a Web site that you are browsing, then the session data may be lost instantly or within a small period of time, depending on how the application is set up. In order to persist state forever, you can use profiles properties.

CHOOSING CLIENT-SIDE OR SERVER-SIDE STATE MANAGEMENT

The type of state management you choose for your web application is determined by various factors. Some of the factors that you would consider are listed in Table 7-5.

Table 7-5

Factors in choosing state management

FACTOR	CLIENT-SIDE STATE MANAGEMENT	SERVER-SIDE STATE MANAGEMENT
Importance of data	Least important data.	Sensitive and critical data.
Server RAM	Can have lower RAM.	Needs to have higher RAM for storing session.
Size of data	Cookies and hidden fields can use data that is very large, which may slow down response time.	Using larger data sizes can slow down application's response time.
Purpose of storing data	For use of data on every page. Query strings can also be passed from page to page.	For use of data across pages and applications.

USING HIDDEN FIELDS

Persisting state across requests in a page by storing it in HTML hidden fields has been common since the days of HTML and classic ASP. In ASP.NET, you can add the `HiddenField` control to a page that submits data through the `post` method to store the state of a single data value. The state data to be persisted is set as the value of the control's `Value` property.

Because the page will be submitted by using the `post` method, the value will be submitted to the server as a part of the form collection as for all other controls.

The hidden field technique is a simple technique that is useful for a small amount of data. Moreover, it is supported on most platforms. However, the data stored in hidden fields can only be a `string` type. Because this technique can be easily manipulated and the data stored in the hidden field can be viewed, the data should not be very sensitive. For example, if you open a web page and select View > Source from the main menu in Internet Explorer, you can see the HTML for the controls on the page, including the hidden field.

To understand the usage of hidden fields, suppose a user is asked to enter his or her name on a web page. Then, the entered value is used to ask the user some questions by addressing the user by his or her name, as shown in Figure 7-18.

Figure 7-18

Hidden fields

The screenshot shows a web page with a header 'TECHNICAL SUPPORTONLINE'. Below the header, there is a list of items. The first item is 'Please enter your name :' followed by an empty input field. The second item is 'Hello <Name>' and the third item is 'Have You installed the product ?'.

In this case, the data is small and not very sensitive from a security perspective. Therefore, it can be stored in a hidden field as follows:

```
<asp:TextBox id="txtName" runat="server"/>
<asp:HiddenField id="hdfName" value="txtName.Text" runat="server"/>
```

In the preceding code, the `Value` attribute is set as the value of the `Text` property of the `TextBox` control named `txtName`. You can retrieve the set value from the `HiddenField` control by using client-side code (JavaScript) thereby eliminating the need for a trip to the server to process the form.

USING VIEW STATE

TAKE NOTE*

In view state, data is stored in key-value pairs. The value of `ViewState` is an instance of the `System.Web.UI.StateBag` class. For information on this class, refer to MSDN.

ASP.NET provides the view state technique for all server controls including the page. This property of the `Control` class (and its derived classes) saves the values of page and control properties that change and need to be persisted between postbacks of a specific page to itself. This data is set as the value of the `ViewState` property as a base64-encoded string. Internally, it is saved in one or more hidden fields between trips to the server.

The following example shows you how to store a value in the `ViewState` property:

```
ViewState.Add("Data", "This is a string value.");
// The following snippet shows how to get the data from the View State.
string data = (string)ViewState["Data"];
```

Some other properties of the `Control` class related to using the `ViewState` property are:

- **EnableViewState:** Disables (`false`) the storage of the `ViewState` or enables (`true`) the storage of the `ViewState`. Note that when enabled for a control, the `ViewState` is stored for the control and its child controls. Check whether the `ViewState` is enabled or disabled by using the `IsViewStateEnabled` property. You should disable `ViewState` for controls for which it may not be required because this technique transfers the data in hidden fields within the page on every request, and the size of the data might affect the performance of the page.

- **HasChildViewState:** Retrieves the value of this property to determine if the child controls of the control have any **ViewState** data saved for them. The value of the property is **Boolean**.
- **ViewStateIgnoresCase:** Retrieves the value of this property to determine if the **ViewState** data stored as the value of the **ViewState** property is handled with case sensitivity. The property value is set to **false** by default indicating the **ViewState** is case sensitive.
- **IsTrackingViewState:** Retrieves the value of this property to determine if the changes to **ViewState** are being tracked and saved for the control.

Note that the **ViewState** data is stored in multiple hidden fields if the size of the data exceeds the maximum size permissible for a hidden field, which is defined by the **MaxPageStateFieldLength** property's value. The value of the property is defined as an integer representing bytes of data. You can set it to a negative integer if you do not want the data to be split across hidden fields. However, in that case some firewalls and proxies may block it based on the maximum limit allowed on them. This property can be set in the web.config file as follows:

```
<pages maxPageStateFieldLength="value" ...>
```

Another important configuration that can be made in the web.config is related to encryption. To improve the security of the data stored in **ViewState**, you can enable encryption for it by making the following setting in the web.config:

```
<configuration>
  <system.web>
    <pages viewStateEncryptionMode="Always"/>
  </system.web>
</configuration>
```

In the preceding code, the value of the attribute **viewStateEncryptionMode** is set as **Always** indicating that the **ViewState** data will always be encrypted. Other values can be **Auto** and **Never**. **Auto**, which is the default value, indicates that the encryption will be in place on request of a control when the **RegisterRequiresViewStateEncryption()** method of the **Page** class is called. The **Never** value ensures that encryption is disabled in all cases. The value of the **viewStateEncryptionMode** attribute can also be set in the **@Page** directive but not in code.

TAKE NOTE *

Another security parameter related to **ViewState** is the **ViewStateUserKey** property of the page class. You can set this property to a unique value for every specific user to identify the user's **ViewState**. For more information on how this property helps, refer to MSDN.

Although ASP.NET stores values in the **ViewState**, you can also explicitly set and read values from the **ViewState**. To read values stored in **ViewState**, you should declare a variable of any type that is serializable and then assign that variable the value of the **ViewState** as shown in the following code:

```
// Declare a variable var of a serializable type
var = (Explicit type cast) ViewState["keyname"];
```

Now, to save values to the **ViewState**, you can use the **Add()** method of **StateBag** class. The method adds a new item to the **ViewState** or updates an existing one. The method accepts the keyname and value as String types. To save the **ViewState**, you can write the statement as:

```
ViewState.Add("keyname", "value")
```

TAKE NOTE*

You can also create a parser for **ViewState**. For such additional information on **ViewState**, refer to MSDN.

For example, suppose you want to pass the number of items viewed by a user on a product catalog page of a Web site. You have tracked and saved this number. You can add it to the **ViewState** and save the value as follows:

```
// Code to track number of items viewed and save in a variable //num
ViewState.Add("itemsViewed", num)
```

If you want to display the count in a label named **lblCount** on the page to which the catalog page submits data, you can read the value from the **ViewState** as follows:

```
lblCount.Text = (string) ViewState["itemsViewed"];
```

Like hidden fields, the view state technique for state management is a client-side technique and therefore, does not require server resources. It is also a simple and basic technique that is used automatically by ASP.NET. Unlike hidden fields, the values stored in state view are not as susceptible to security threats as value stored in hidden fields because the values are encoded before being stored. However, they are not completely secure and can be tampered with because they use hidden fields as underlying storage. The main disadvantage of view state is that the view state needs to be passed back and forth between requests and can affect the performance if the data is heavy. In addition, it is suitable only for page-level state scope.

USING CONTROL STATE

View state can be disabled for controls or at the page level. However, at times controls may need to persist crucial data across requests. In this case, you can use control state instead of view state. However unlike view state, control state implementation needs additional programming to be accomplished by the developers.

Control state is stored in the **ControlState** property of the **PageStatePersister** class. The property value is an **Object** type. It can be used to store data that is required across requests of the same page. Like view state, control state is also stored in hidden fields.

TAKE NOTE*

Control state is used often in the case of custom control. In order to do so, the custom control must override the **OnInit()** method of the **Control** class and then invoke **RegisterRequiresControlState()** in the method. Thereafter, the custom control should override the **LoadcontrolState()** and **SaveControlState()** methods.

TAKE NOTE*

If a query string key name or value has space or ampersand characters, you can respectively use %20 and %26 for them.

USING QUERY STRING

Another client-side state management technique is passing data between pages as a query string. A query string is an expression appended to a URL that has the following format:

?key1=value1&key2=value2...

However, unlike the techniques discussed previously, query strings cannot be used for pages that use the **HTTP POST** method for submitting data to the server. They only work when **HTTP GET** is used.

To use query strings for state management, first you need to form the URL of the page to which data needs to be submitted from a page by appending a query string. The following code example shows how a query string can be appended to a URL:

```
// Code in Login.aspx.cs

string newUrl = "~/default.aspx?";
newUrl += "Name=" + txtName.Text + "&";
newUrl += "Pass=" + txtPwd.Text;
Response.Redirect(newUrl );
```

In the preceding code, the URL with query string is formed by appending the values of username and password entered in two text boxes of the Login page. The page is then redirected to default.aspx passing the values of the username and password as a query string for validation.

In the default.aspx page, these values passed in the query string can be retrieved by using the `Request` object as follows:

```
string username = Request.QueryString["Name"];
string pwd = Request.QueryString["Pass"];
// Code that validates the username and pwd variables' values
```

TAKE NOTE*

There are various query string builder tools also available that make the process of forming query strings simpler and less prone to errors.

Query strings are a simple mechanism for state management at the client level. However, they are not suitable when you want to pass sensitive information such as password, credit card numbers, and so on. This is because any user can see the information being passed in the address bar of the browser. Even if you encode and send data, it is still prone to attacks from malicious users. Therefore, it is advisable to avoid using query strings for sensitive information. In addition, forming query strings manually is a time-consuming and error-prone process that should only be used when the number of values to be passed is small and simple. Some browsers impose a limit on the number of characters in the query string.

USING COOKIES

Cookies are small text files stored on the client side by the web application. They contain state data that can be sent back to the server across multiple requests and even across different sessions of a client. They help to identify and remember a user. To understand the use of cookies consider a scenario. Suppose you visit a Web site and on your visit you are asked to select a preferred language and home page customization options, such as background color, foreground color, picture to be displayed on the home page, and so on. This information of a particular user can be stored in a text file and saved on the client machine. When the user later visits the Web site using the same client machine, along with the request for the home page of the Web site, the data stored in the cookie will be sent to the server by the browser. The server will read the data and render the customized home page for the user.

Cookies can be of two types: persistent or temporary. Persistent cookies stay on the client machine until they are explicitly deleted by the user or server. However, temporary cookies have an expiration time associated with them. Moreover, cookies can be single valued or multiple valued. Multiple-valued cookies have multiple name-value pairs that are called subkeys of the cookie. For example, you can store the preferred language, background color, and foreground color all in a single cookie as subkeys.

Let us understand how ASP.NET supports cookies. A cookie is represented through the `System.Web.HttpCookie` class. To create a cookie, you can instantiate this class by using its overloaded constructor:

- **HttpCookie(string name)**: Creates a cookie that has the name passed as an argument.
- **HttpCookie(string name, string val)**: Creates a cookie that has the name passed as the first argument and assigns it the value passed in the second argument.

Some of the commonly used properties of this class are:

- **Name**: Defines a name for the cookie. It is set when you instantiate the `HttpCookie` class.
- **Value**: Defines the value for the cookie. It can be set by using the second overload of the constructor.
- **Values**: Retrieves a `NameValuePairCollection` type representing multiple values in a cookie.
- **Expires**: Sets the expiration date and time for the cookie as a `DateTime` type value. If you do not set this property, the cookie is understood to be temporary and is not stored on a hard disk on the client machine. It is kept in memory while the session is going on and removed from memory when the session ends.
- **Domain**: Limits the accessibility of cookies to pages in a specific domain or subdomain. Further, you can also limit the folder by setting the `Path` property. Then the cookie will be associated with pages within that folder. Set the domain name and path as string values.
- **HasKeys**: Determines whether a cookie has multiple values. The property will be set to `true` in that case. By default, it is set to `false`.

MORE INFORMATION

For a complete list of members of the `System.Web.HttpCookie` class, refer to the `HttpCookie` Members section on MSDN.

Cookies are manipulated using the `Cookies` property of the `HttpResponse` and `HttpRequest` objects. This property value is an `HttpCookieCollection` type, which represents the current collection of cookies. The former object allows you to work with the collection of cookies to be added to the client machine, and the latter object allows you to work with the collection of cookies to be retrieved from the client machine.

To write cookies, set their properties, and send them to the client, you can use the `HttpResponse` object as follows:

```
HttpCookie langCookie = new HttpCookie("langInfo");
langCookie.Value = "English";
langCookie.Expires = DateTime.Now.AddDays(2);
Response.Cookies.Add(langCookie);
```

In the preceding code, a cookie named `langInfo` is added to the collection of cookies to be sent to the client with the value set as `English`. The expiration is also set as two days from the current date. A simpler way to add cookies is:

```
Response.Cookies["langInfo"].Value = "English";
Response.Cookies["langInfo"].Expires = DateTime.Now.AddDays(2);
```

To create a single cookie storing two values, preferred language and background color, use the following code:

```
// Response is an HttpResponse object
HttpCookie custCookie = new HttpCookie("customizeInfo");
custCookie.Value ["langInfo"] = "English";
custCookie.Value ["bgcolorInfo"] = "Red";
custCookie.Expires = DateTime.Now.AddDays(2);
Response.Cookies.Add(custCookie);
```

or

```
Response.Cookies["custCookie"]["langInfo"].Value = "English";
Response.Cookies["custCookie"]["bgcolorInfo"].Value = "Red";
Response.Cookies["custCookie"].Expires = DateTime.Now.AddDays(2);
```

To modify cookie or subkey values, set the cookie or subkey to a new value. You can explicitly remove a subkey, such as `langInfo`, as follows:

```
custCookie.Values.Remove(langInfo);
```

To read a cookie sent by the browser, you can use the `HttpRequest` object:

```
// Request is an HttpRequest object
if(Request.Cookies["langInfo"] != null)
{
    HttpCookie langCookie = Request.Cookies["langInfo"];
    lblLang.Text = langCookie.Value;
}
```

Alternatively, you can use the following code:

```
if(Request.Cookies["langInfo"] != null)
    lblLang.Text = Request.Cookies["langInfo"].Value;
```

In the preceding code first the value of the cookie is checked to see if it exists, and then, the cookie is read by using the `Cookies` property of the `Request` object.

To ensure that the cookies have not been tampered with, you can use the `Server.HtmlEncode()` method with cookies and query strings in this manner.

```
lblLang.Text = Server.HtmlEncode(Request.Cookies["langInfo"].Value);
```

To delete a cookie, you should create a new cookie with the same name as the cookie to be deleted. Then, you should set its expiration date in the past. The cookie will be discarded by the browser when this new cookie is sent to it.

Cookies are also a simple and useful technique for identifying users. The primary problem with cookies is that cookies can also be disabled explicitly on browsers. In addition, the size of cookies supported by most browsers is limited to 4 kilobytes. Moreover, the number of cookies stored by a Web site is also limited to 20 on most browsers.

USING SESSIONS

Because cookies are used to identify a user on the client side, session variables are used to identify users on the server side. Session state refers to a collection of variables called session variables that are used to store data as key-value pairs, accessible to all pages requested by the current session. A session refers to all the connections and requests that a user makes with the server while browsing through a Web site in one go. For example, if you open a Web site, browse through its Home page, Catalog page, and then close all pages of the application—it is considered as one session. The next time you open any page of the Web site, a new session starts.

Technically, a session state is represented by the `System.Web.SessionState.HttpSessionState` class. A session is uniquely identified with the help of a session ID. All page requests made using the same session ID belong to one session. ASP.NET generates unique session IDs for every new request with which no session ID is associated. If a session ID has been associated with the request from a user, the same session ID is used for all requests from that user. The session ID is stored on the client-side in a cookie or passed in the query string. You can retrieve the session ID by using the `SessionID` property of the `HttpSessionState` class. This class has other useful properties associated with a session, such as:

- **CookieMode:** Determines whether the session ID is configured to be stored in cookies or passed as a query string by using this property. The value of the property can be any member of the `HttpCookieMode` enumeration, which includes `AutoDetect`, `UseDeviceProfile`, `UseCookie`, and `UseUri`.
- **IsCookieless:** Determines whether session ID is stored in a cookie or passed in query string.
- **Mode:** Retrieves the storage mode, such as SQL Server or custom data store, used by the session state using this property. The value of this property can be any member of the `SessionStateMode` enumeration, which include `Off`, `InProc`, `StateServer`, `SQLServer`, and `Custom`. The value `Off` disables session state. The value `SQLServer` is used to store data in SQL Server and requires further programming to implement the storage. The value `InProc` stores the data in memory and is a fast and the default way to store session data. The value `StateServer` stores the data in ASP.NET State Service, which preserves the session state even when the application is restarted (unlike `InProc`). The `Custom` option is used when you want to implement custom storage.
- **Timeout:** Sets the session expiration time in minutes. A session can expire if a request for the session ID is received after the specified time is elapsed since the last request from the same session ID. The default value is 20.

The preceding properties can be set as attributes of the `sessionState` element in the web.config as follows:

```
<configuration>
  <system.web>
    <sessionState
      cookieless="true" mode="SQLServer"
      timeout="60" .../>
    </system.web>
  </configuration>
```

MORE INFORMATION

For more information on cookies, refer to the `ASP.NET Cookies Overview` section on MSDN.

MORE INFORMATION

For a complete list of properties and methods of the `System.Web.SessionState` and `HttpSessionState` class, refer to MSDN.

Next, let us discuss how session variables are stored.

Session variables are stored as key-value pairs in an instance of the `System.Web.SessionState.SessionStateItemCollection` class. You can access the current session variables through the `Session` property of the `Page` class. To write a session variable, consider the following code example:

```
Session["bgcolorState"] = txtColor.Text;
```

In the preceding code, a key named `bgcolorState` is added as a session variable to the collection, and its value is set as the value of the `Text` property of a text box named `txtColor`.

To read values, you need to explicitly type case them because they are stored in the collection as `Object` type. The following code shows you how to read the `bgcolorState` session variable, defined previously:

```
if (Session["bgcolorState"] != null)
string color = (string) Session["bgcolorState"];
```

USING APPLICATION STATE

If you want to store data that has application-level state scope so that it is accessible to all pages of the application and applicable for all users and sessions, then you can persist the state by using the application state. This is a server-side state management technique that uses the `HttpApplicationState` class. The data stored using application state is stored in memory on the server and can be quickly retrieved.

The `HttpApplicationState` class instance corresponds to a specific web application and is created when a user requests a URL of the web application for the first time. It stores the data as key-value pairs. To access the instance associated with a web application, the `Application` property of the current page or the `HttpContext` class is used. To write data to application state, you can use the following code:

```
Application.Lock()
Application["FooterText"] = "Copyright 2009";
Application.UnLock()
```

In the preceding code, first the application state is locked by calling the `Lock()` method so that only the current thread can write to the application state. Then a key named `FooterText` is added, and its value is set as `Copyright 2009`. Finally the application state is unlocked.

In application state, the data is stored in the memory or RAM.

To read a value from the application state, consider the following code:

```
if (Application["FooterText"] != null)
{
    String footerVal = Application["FooterText"].ToString();
}
```

In the preceding code, first the existence of the pair is checked in the application state. Next, the value is retrieved and type cast.

You can set application state values in the `Application_Start` event handler. You can clear to change the values in the `Application_End` event handler. This technique is simple in implementation because it is managed by the server but it uses server resources to store data. However, it has the advantage of providing application level scope for crucial state data.

MORE INFORMATION

Session management in ASP.NET also allows you to raise events specific to sessions. For more detailed information on sessions, refer to MSDN.

CERTIFICATION READY?

Configure session state by using Microsoft SQL Server, State Server, or InProc.

1.4

TAKE NOTE *

Locking is optional but you should use it to prevent corruption of data.

TAKE NOTE *

USING PROFILE PROPERTIES

Sometimes you need to store user preferences permanently on the server, even after the session has ended. In this case, session state does not suffice and you can use profile properties.

In ASP.NET a profile is a repository of any type of data associated with a specific user. A profile can uniquely identify a user and manage the underlying storage and retrieval tasks on its own without the need for you to design a database and write code to manipulate it.



USE THE PROFILE FEATURE

To use the profile feature, perform the following steps:

1. Add profile properties in the configuration file. For example, you can add a property for the preferred background color to be used for customization as follows:

```
profile>
  <properties>
    <add name="bgColor" />
  </properties>
</profile>
```

2. Set the value for the defined properties in the code by using the `Profile` property exposed by the current `HttpContext` and accessed through the pages in the web application. This is done as follows:

```
Profile.bgColor = txtColor.Text;
```

The value, when set, is automatically associated with the current user and does not require any coding.

3. Retrieve the value on subsequent visits of the user to the Web site. You should type case the value appropriately as they are stored as `Object` type. This is shown in the following example:

```
string color = (string)( Profile.bgColor );
```

CERTIFICATION READY?

Implement session state, view state, control state, cookies, cache, or application state.

7.5

TAKE NOTE *

By default, the `SqlProfileProvider` class is used to store and manage the `Profile` properties. However, you can use a different provider also. For detailed information on profile providers, refer to MSDN.



USE THE HIDDENFIELD CONTROL

1. To store the time of postback in the `HiddenField` control, when the user posts back the data, write the following code:

```
<form id="form1" runat="server">
<div>
  <asp:Label ID="Label1" Text="Enter Your Name:" runat="server" />&nbsp;
  <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
  <br />
  <asp:Label ID="Label2" Text="Enter eMailAddress" runat="server"></asp:Label>&nbsp;
  <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
  <br />
  <asp:Button ID="Button1" runat="server" Text="Submit" onclick="Button1_Click" />
  <asp:HiddenField ID="HiddenField1" Visible="false" runat="server" />
</div>
</form>
```

2. Write the following code in the click event handler of the button:

```
this.HiddenField1.Value = DateTime.Now.ToString();
Response.Write(this.HiddenField1.Value);
```

You can run this code by pressing F5. On the form that is displayed, you can now click the Submit button. The date and time when you clicked the button will be stored in the hidden field and displayed.

SKILL SUMMARY

This lesson discusses the three most important concepts in Web application development: validation, navigation, and state management.

ASP.NET Validation Framework consists of various validator controls that can be used to validate the user inputs such as RequiredFieldValidator, RangeValidator, CompareValidator, RegularExpressionValidator, and CustomValidator. You learned how and when to use these validator controls.

You have also learned about various navigation controls that allow users to navigate from one page to another. You learned about the SiteMap, Menu, and TreeView controls in this regard. This lesson explained how to use the SiteMap control to create a web.sitemap file. You learned that the Menu control allows you to create vertical and horizontal menus and that you can use menus with a Site Map. The TreeView control is used to display hierarchical data. You also learned how to format the TreeView control using various methods. Finally, this lesson discussed URL remapping, which you could use to display different pages than what the user requested.

Finally, this lesson described state management and the different techniques for state management. You learned about client-side and server-side state management and the techniques for implementing client-side and server-side state management, such as ViewState, control state, application state, cookies, query string, and hidden fields. The ViewState property can be used to retain data during postback. Cookies and query strings can also be used to store and retrieve the data. You learned about the different session states and how you can use the Application object to store and read data from it.

For the certification examination:

- Understand the validation framework and implement client-side and server-side validation.
- Configure providers.
- Implement data-bound controls.
- Configure session state by using Microsoft SQL Server, State Server, or InProc.
- Understand the state management techniques and implement session state, view state, control state, cookies, cache, or application state.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. All validator controls are inherited from the _____ class.
2. The _____ control ensures that the user has entered or selected a value in the designated field and displays an appropriate error message if the value is not entered.
3. The _____ property allows you to set up a custom validator with a client-side function.
4. The _____ class serves as a representation of the navigation structure for a site.
5. _____ is a helper class that provides the methods used by both URL rewriting HTTP module and HTTP handler.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---|---|
| T | F 1. All compare validators are inherited from the <code>BaseValidator</code> class. |
| T | F 2. Implementing client-side validation is difficult to achieve but secure. |
| T | F 3. You can use the <code>TreeView</code> control to display hierarchical data. |
| T | F 4. The <code>VirtualPathProvider</code> class can be used with global.asax pages. |
| T | F 5. You can implement state management at the session level and the application level |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. Which of the following navigation controls presents navigational information in a “bread-crumb” format?
 - a. Menu
 - b. `TreeView`
 - c. `SitemapPath`
 - d. `Sitemap`
2. Which of the following is part of client-side state management?
 - a. Application state
 - b. Session state
 - c. View State
 - d. Profile
3. What is the default value of hidden fields?
 - a. null
 - b. true
 - c. false
 - d. none
4. With which of the following can you use the `VirtualPathProvider` class?
 - a. Global.asax
 - b. Web.config
 - c. Master pages
 - d. Bin folder

Review Questions

1. Why is navigation between nested pages difficult to implement?
2. What is the advantage of using ASP.NET’s navigation and site map support?

■ Case Scenarios

Scenario 7-1: Designing Site Navigation

You are creating a large application for a banking organization. The application will have multiple subsites within the site for different categories of banking products. Using each subsite, users can obtain information regarding credit cards, debit cards, online bank accounts, mutual funds, and so on. You foresee that it will be difficult for the user to locate a specific service from this huge application. How will you solve this problem?

Scenario 7-2: Using State Management

You are developing a Web site for gaming. Here, multiple users can log on and play the game. You need to ensure that the next time the user logs on he can resume the game from the point where he left. How will you ensure this?

Scenario 7-3: Using URL Remapping

You are working on enhancements to an already-developed web application. This involves creating new web pages, enhancing the existing web pages, and removing some pages. How will you ensure that the references to the old pages will get redirected to the newly developed page?



Workplace Ready

Understanding Validation

Consider that you are developing a Web site for an easy bill-pay site. The Web site should allow customers to register and pay all utility bills using their credit cards. How will you implement the validations?

Working with Globalization, Localization, and Accessibility

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing globalization and localization.	Implement globalization and accessibility.	7.3
Configuring accessibility.	Implement globalization and accessibility.	7.3

KEY TERMS

accessibility**explicit localization****globalization****implicit localization****localization**

Suppose you have developed a Web site for a leading enterprise that retails electronic and home appliances in over 50 countries. You have designed an aesthetically appealing and feature-rich web shopping application for this retailer. However, the content on the Web site is displayed in English, and the product pricing is only displayed in dollars. In such a case, your web application is likely to be less popular with users from non-English-speaking regions of the world such as China and Japan. Moreover, displaying all pricing in dollars is a drawback for the Web site because users will not want to spend time trying to calculate amounts in their own currency. Therefore, it is imperative that your web applications cater to regional media and content if you want to truly leverage the power and reach of the World Wide Web. This may pose a challenge for developers to design content that can reach users with varied language, culture, and formatting standards. In such a scenario, **globalization**, **localization**, and **accessibility** techniques are useful in making your applications usable for a wider audience.

■ Introducing Globalization and Localization



The globalization, localization, and accessibility features supported by the .NET Framework allow developers to handle regional media and content in web applications that need to be targeted to a global audience. Globalization allows you to develop a web application that can be customized according to the needs and preferences of people who live in different parts of the world. A global application is flexible enough to input, process, and output different languages, number and data formats, currencies, text direction, text casing, and date/time systems according to the preferences of the user accessing the application.

In order to implement globalization in a web application, you need to first determine the elements of the application that would change based on the preferences of the user accessing the application. These elements, called localizable resources, need to be separated out. After identifying and separating localizable resources, you need to code the rest of application so that it is neutral to any specific language or data formats specific to a particular region. This is called internationalizing your code.

Internationalization alone does not make an application global. The application also needs to be localized so that it can run successfully in different languages and support different data formats. The task of separating out the localizable resources is the beginning of localization. Localization refers to translating and creating content and resources so that they become specific to particular languages and data formats that you want your web application to support. For example, the description on the About Us page of a company's Web site, all controls captions on a page, and the titles of the page, are all localizable resources that would change if the user is from a different region or has different preferences.

In addition to globalization and localization, another important aspect that increases the user base of a Web site is the support for accessibility features on that Web site. Accessibility features make an application easy to use and more accessible for users, particularly those with disabilities. For example, support for varying input devices, display of alternate text if an image is not loaded, and setting the tab order of controls on a form are examples of accessibility features.

In this lesson, you will learn how globalization, localization, and accessibility features are implemented in ASP.NET applications.

Implementing Globalization and Localization

You can use local and global resources provided by ASP.NET to develop the Web site and to create a flexible web page structure to support different languages. You can then set the language and formatting of the web page programmatically based on the user's preference.

Consider that you need to maintain a large Web site that allows the users to select their language preference and view the site in that language. You need to develop that Web site so that some pages are available for all users in the default language; additionally, users should be able to customize the language displayed on other pages. In addition, you need to update this database frequently.

You can display a web page in different languages by translating the text on screen as well as on the controls on the web page. This requires you to rewrite text for all controls within the code where the user selects the preferred language. This is a time-consuming task for Web sites that are large or that require frequent updates. In addition, to perform this task, translators need to be familiar with programming so that they know where to insert the translated text. If the translator inserts the code in the wrong location, it could lead to an error on the page.

ASP.NET uses resource files that provide a much simpler process to support multiple languages. Using Visual Studio 2008, you can generate XML resource files containing text for the controls on the web page. The text is saved in name-value pairs where each name matches a named element on the web page. When you program, you use a name for the string rather than its actual content and .NET Framework maps the name to the appropriate culture depending on the user's settings. This eliminates the need for the translator to work on the source code, thus eliminating the risk of errors due to incorrect code.

When a user visits your Web site, the web browser automatically displays a web page that best meets the language and culture options set in the browser. If the user wants to view the web page in a different language, the web browser provides the user with an option for language preference. Based on the user's selection, ASP.NET automatically displays text using

that language's resources, if available. Now, let us discuss how you can use local and global resources in your web application.

The implementation of globalization and localization in ASP.NET is based on two core concepts:

- Culture
- Resource files

DEFINING THE SUPPORTED CULTURE

The preferences of users accessing web applications vary based on the following factors:

- Language
- Geographical region, which affects the language. For example, United Kingdom English and United States English are two different languages if you consider the regions
- Data formats, such as format of date, calendar, currency, and numbers

The preceding factors are collectively called the culture and define the preferences of the user differently for different users based on which web application is run.

All cultures to be supported by a global web application should be defined for it during development. This is accomplished by using the classes and enumerations defined in the `System.Globalization` namespace.

An important class for defining the culture-related information is the `CultureInfo` class of the `System.Globalization` namespace. An instance of this class represents a culture. It is created by invoking the overloaded constructor of the class. The culture is created on the basis of the culture name passed to the overloaded forms of the constructor as a string or as an integer (culture code). In the .NET Framework, culture names are defined in the format `<language>_<region>` where `<language>` is a two-letter, lowercase code representing a language name, and `region` is a two-letter, uppercase code representing a geographical region; for example, U.S. English is represented as `en-US`, which stands for the U.S. dialect of English.

Two other overloaded forms of the constructor accept a Boolean value indicating whether custom values set by a Windows user should be used. All cultures and locales defined in Windows can be configured from the Control Panel, as shown in Figure 8-1.

Figure 8-1

Configuring cultures and locales



If the value of the second argument in the overloaded forms of the constructors is set to **true**, these settings are considered when defining the culture, otherwise they are ignored.

Three important properties of the **CultureInfo** class are:

- **CurrentCulture**: Retrieves the **CultureInfo** instance representing the culture defined for the current thread. It defines all the culture-dependent data format settings for the current thread.
- **CurrentUICulture**: Retrieves the **CultureInfo** instance representing the culture based on which the associated resource files will be loaded for the current thread to run the application according to the culture. This means that if the retrieved value indicates a German culture, the resource files associated with German content will be loaded so that the UI elements of Web site associated with the loaded resources can be displayed for a German user.
- **CultureTypes**: Retrieves the culture types represented by the **CultureInfo** instance. In the .NET Framework, the types of cultures are defined in the **CultureTypes** enumeration. Some commonly used values of this enumeration are:
 - **NeutralCultures**: Refers to cultures that are associated with only a language but not a region. They are expressed only using the two-letter language code. The region code is omitted. For example, **fr** can be the name of a neutral culture related to the French language. You can check whether a **CultureInfo** instance represents neutral culture by using the **IsNeutralCulture** property of the class.
 - **SpecificCultures**: Refers to cultures that are specifically associated with a language and a region. For example, **el-GR** is a specific culture associated with the Greek language of Greece.
 - **InstalledWin32Cultures**: Refers to all cultures installed on Windows.
 - **UserCustomCultures**: Refers to custom cultures created by users. You can create a custom culture by using the **CultureAndRegionInfoBuilder** class and registering it. You can give the custom culture any name, unlike the predefined culture names.
 - **AllCultures**: Refers to all neutral, specific cultures and custom cultures installed on Windows.

Other important properties of the class associated with the **CultureInfo** instance are those that return data formats related to the culture, such as:

- **Calendar**: Retrieves an instance of the **System.Globalization.Calendar** class, which defines the default calendar for the culture—specific or invariant only.
- **DateTimeFormat**: Retrieves an instance of the **System.Globalization.DateTimeFormatInfo** class, which describes the format for displaying date and time for the culture—specific or invariant only.
- **NumberFormat**: Retrieves an instance of the **System.Globalization.NumberFormatInfo** class, which describes the number system, such as currency, percentage, and number display used for the culture—specific or invariant only.
- **TextInfo**: Retrieves an instance of the **System.Globalization.TextInfo** class, which describes the writing system, such as casing and text separators used for the culture.

In addition to specific and neutral, a third type of culture is the invariant culture—a culture that is actually not associated with any culture. It only represents the English language. However, it is not a neutral culture and is named by using an empty string or only **en**. It can be retrieved by using the **InvariantCulture** property of the **CultureInfo** class. It is used for data that is not intended directly for the user. For example, consider the case of the Web site of an electronic and home appliances retailer. If the Web site accepts feedback from users on their experiences with products or services and saves it to a text file for the perusal of company management staff, then the invariant culture can be used for the text saved to the file so that it is not sensitive toward any culture and can be used as desired. For example, it can be viewed in English by management staff or displayed on a web page for users by interpreting it to their specific cultures.

TAKE NOTE*

Some useful methods of the `CultureInfo` class are:

- **GetCultures(cultureType)**: Returns the list of cultures, as an array of `CultureInfo` objects. The returned value represents the cultures of the types passed as an argument to the method. The argument can be a bitwise combination of values of the `CultureTypes` enumeration, as desired.
- **GetCultureInfo()**: Returns a cached, read-only instance of the `CultureInfo` class. It retrieves a `CultureInfo` instance faster than by using a constructor. Therefore, it is useful when multiple instances of a culture need to be created.
- **CreateSpecificCulture()**: Returns a `CultureInfo` instance representing a specific culture associated with the name passed in the argument. Even if you specify a neutral culture name in the argument, the instance returned would represent a specific culture associated with the name passed in the argument. If an empty string is passed, the returned instance represents an invariant culture.

MORE INFORMATION

For detailed information on the members of the `System.Globalization` namespace, refer to MSDN.

TAKE NOTE *

Only specific cultures can be set for value of the `Culture` property.

Now that you are aware of the basics of the `CultureInfo` class, let's understand how culture-related settings are accomplished in an application.

By default, when an application thread is started and no culture is set for it, the culture settings are picked from the default Windows settings for language and culture on the web server. However, you can change them per user preferences declaratively or programmatically.

To set the culture declaratively at the application level for all pages, you can make the following entry in the `web.config` file and save it in the root folder of the application:

```
<globalization culture = "fr-FR" uiCulture="en">
```

In the preceding declaration, all data formats for all pages will use the French culture, whereas the English resources will be loaded for the UI culture. This means that dates, calendars, currency, and so on will be according to French culture but the context, text on controls, and titles will be in English. Here, you should assume that you have already set the resources.

You can set specific culture values for a page declaratively as follows:

```
<% @Page UICulture="en" Culture="fr-FR"%>
```

Programmatically, you can set the culture values for a page by overriding the `InitializeCulture()` method of the web page to define the two values as follows:

```
protected override void InitializeCulture()
{
    this.Culture = "<specific culture name>";
    this.UICulture = "<culture name>";
}
```

TAKE NOTE *

You perform the culture-related settings in the `InitializeCulture()` method of the page because the method is executed early in the life cycle of the page, even before any controls are generated.

The preferred method for setting preceding values programmatically is at the thread level.

To define the UI culture value, create an instance of the `CultureInfo` class by using any of its overloaded constructors and set the value of the `CurrentUICulture` property of the current thread to the `CultureInfo` instance, as follows:

```
Thread.CurrentThread.CurrentUICulture = new
CultureInfo("<culture name>");
```

To define the culture value, which only accepts specific culture values, you can use the constructor as explained earlier but ensure that you pass a specific culture value to it. Another option is to invoke the `CreateSpecificCulture()` method of the `CultureInfo` class, as follows:

```
Thread.CurrentThread.CurrentCulture =
CultureInfo.CreateSpecificCulture("<specific culture name>");
```

For the preceding values, the culture name passed as an argument can either be hard coded or dynamically obtained, which is the more flexible and preferred way in most cases. You can dynamically obtain the culture values in two ways:

- **Use the browser settings:** You can pick the culture values from the settings defined in the browser as shown in Figure 8-2 for Internet Explorer.

Figure 8-2

Culture settings in Internet Explorer

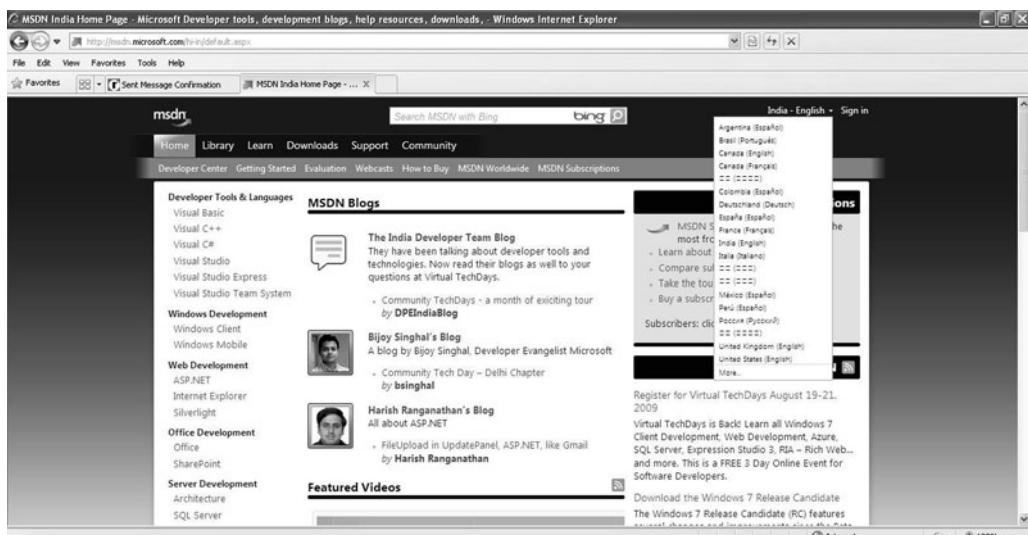


The preceding window opens from Internet Explorer by selecting Tools > Internet Options from the main menu, and then, clicking the Language button in the General tab. You can change the culture settings from here for the purpose of testing your applications. Also, note that to use the browser settings, you must set the `enableClientBasedCulture` attribute to `true`, and then, set the culture and UI culture to a value of `auto` in the `web.config` file, in the `@Page` directive, or in the program.

- **Let the user choose:** The more common and preferred way to change the culture value for a user is by asking the user to choose an option. This is because the option is more flexible and prevents errors or problems due to blocked browser settings or missing culture in the culture set supported for a browser. Many Web sites allow users to select the region from a drop-down list as shown in Figure 8-3.

Figure 8-3

Setting culture based on user selection



Based on the selected region, the culture and UI culture values are set for the current thread, and further web pages are displayed accordingly.

How is the set culture value used? The `CurrentCulture` value is used to format all data in the code. For example, you can format currency value per the current culture as follows:

```
CultureInfo frCulture = CultureInfo.CurrentCulture;
double price = 678.56;
string frenchPrice = price.ToString("C", frCulture);
```

However, merely setting the culture and UI culture does not automatically translate all content from one culture to another. The translation of content, such as text and images displayed on the UI is required because the .NET Framework does not act as a translator of content. It only handles the data formats related to a culture and helps developers to map the translated content to the culture. However, the translation of the content has to be done explicitly. Managing and mapping the translated content can be done in a simple manner with the help of resource files.

USING RESOURCE FILES

Resource files store translated values of localizable resources. For example, if you want to change the Label controls on a web page according to the culture values, the names of the Label controls and their respective values for specific cultures will be stored in resource files. Resource files are written in XML and contain name-value pairs, in which the name represents the name of a localizable resource, and the value refers to the value of the resource in a specific locale and/or culture.

TAKE NOTE *

The default base files are used when no current UI culture is explicitly specified or if the specified value does not match any of the supported UI cultures for which resource files exist.

You can generate a resource file for every neutral culture (only language) or for every specific culture (language and region). Resource files are named `<base name>.<culture>.resx`. For example, you can create a resource file containing values of localizable resources for the U.S. English culture with the name `Home.aspx.en-US.resx`. The resource file loaded for the application depends on the value currently set for the UI culture. So, if the UI culture is U.S. English, the `Home.aspx.en-US.resx` file will be loaded to refer to the values of the resources defined in it. Alternatively, if the value of the current UI culture is Danish, the `Home.aspx.ds.resx` file will be loaded. Note that you should create a base resource file that will be used as the default if no specific value is set for the current UI culture. The name of the base resource file is specified as `<base name>.resx`, for example, `home.aspx.resx`. Ideally, you should create the default file, and then, you can create, edit, and rename the copy for specific cultures.

Resource files do not require manual compilation and are compiled by the .NET Framework into assemblies called satellite assemblies at runtime.

Before you learn to create and load resource files, let us identify the types of resource files, based on which the scope, naming, and location of resource files are defined.

Resources or resource files are of two types:

- **Local:** Provides different languages and formatting support for a particular web page (`.aspx`), a master page (`.master`), or a specific user control (`.ascx`). All local resources should be stored in the `App_LocalResources` subfolder of a folder in the web application. There can be `App_LocalResources` subfolders in more than one folder of the web application. Storing in this specific path is important because every page in your web application might have local resources. For local resource files, the `<base name>` is the name of the page for which the resource file has been created, as explained in the preceding example for the `Home.aspx` page.
- **Global:** Applicable at the application level. They can be accessed by all code in an application and multiple controls can be associated with the same resource file. For example, say the electronic and home appliances retailer wants to display his tag line on every page of the Web site. In such a scenario, the developer can associate the Label controls used for the tag line on all web pages with a global resource file. This global resource file will store the tag line, which would be the value of the Label control's `Text` property, in different languages. Unlike local resources, all global resources must be stored in a folder named `App_GlobalResources`, which is located

at the root location of the web application. The <base name> in global resource file names should not be the name of any page—it can be any meaningful name, such as `RetailGlobal.fr-FR.resx`.

CREATING RESOURCE FILES

You can create resource files by any of the following methods:

- Manually by using the Add New Item option in Visual Studio 2008, and then, using the Resources Editor.
- By using the Generate Local Resource option on the Tools menu of Visual Studio 2008 in the Design view.



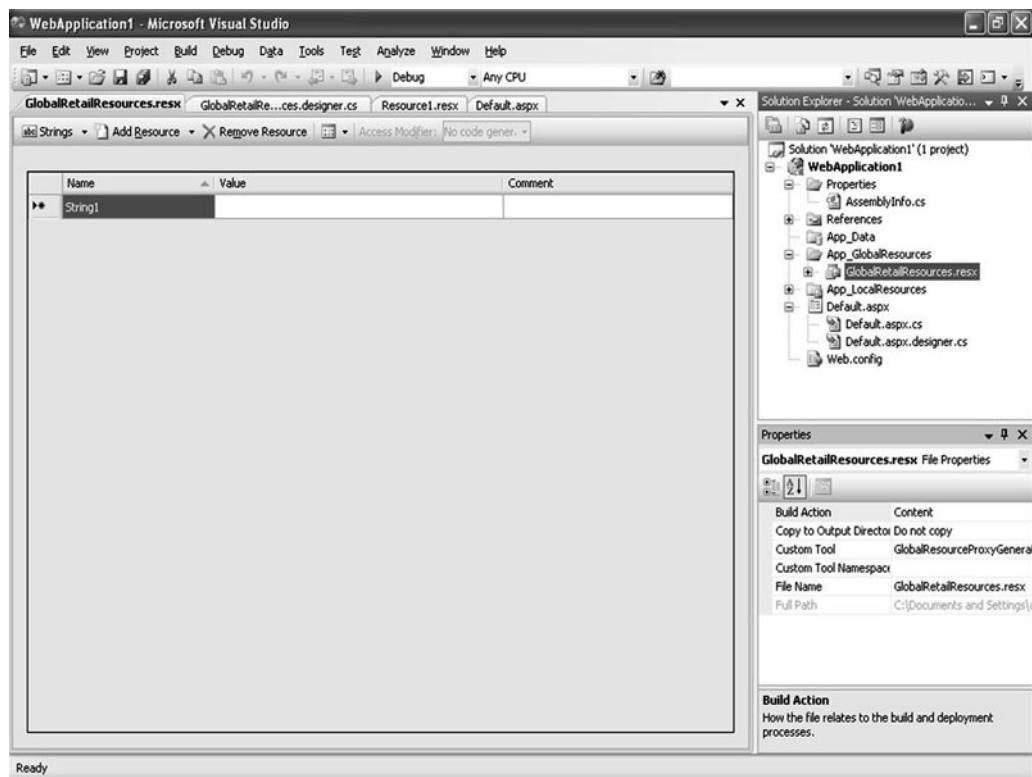
CREATE RESOURCE FILES MANUALLY

To create a resource file manually:

1. Ensure that the appropriate folder for local or global resource files has been created.
2. In the Solution Explorer, right click the folder, and select Add > New Item from the shortcut menu. The Add New Item dialog box will open.
3. In the Add New Item dialog box, select the Resources option from the Visual Studio Installed Templates list, specify a name for the resource file in the Name field, and click the Add button. The Resource Editor will open as shown in Figure 8-4.

Figure 8-4

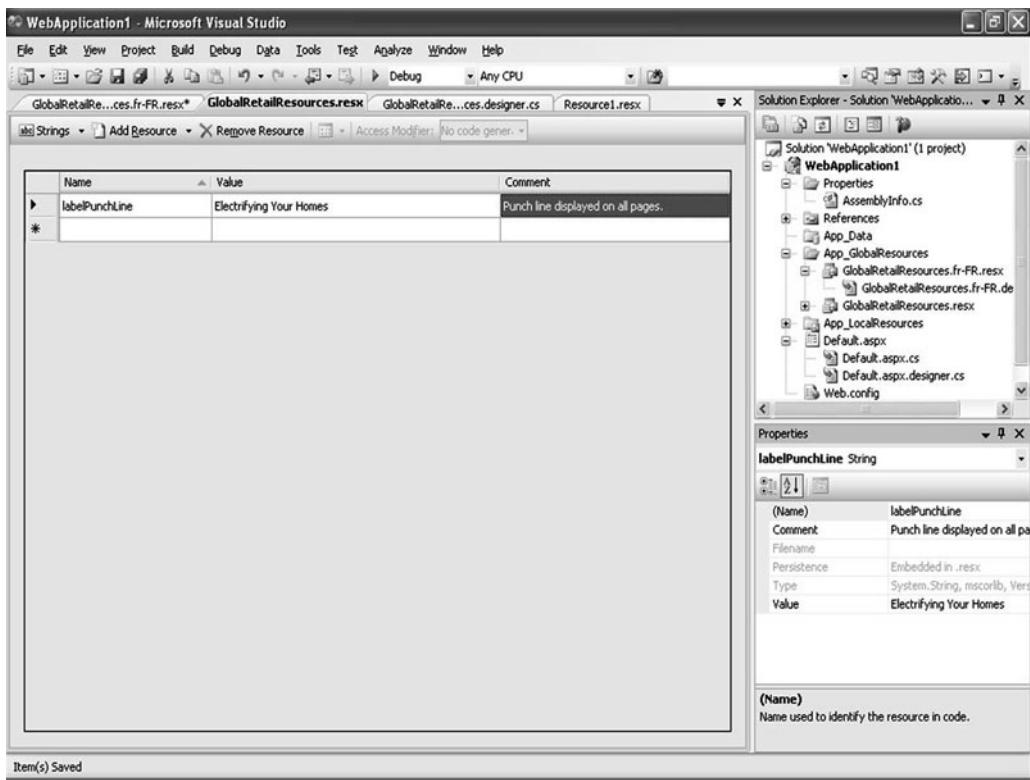
The Resource Editor



4. In the Resource Editor, specify the names of the keys and their values, and save the file. If you consider the tagline example for the electronic and home appliances retailer, Figure 8-5 shows the entries in the Resource Editor for the global resource files.

Figure 8-5

Resource Editor entries

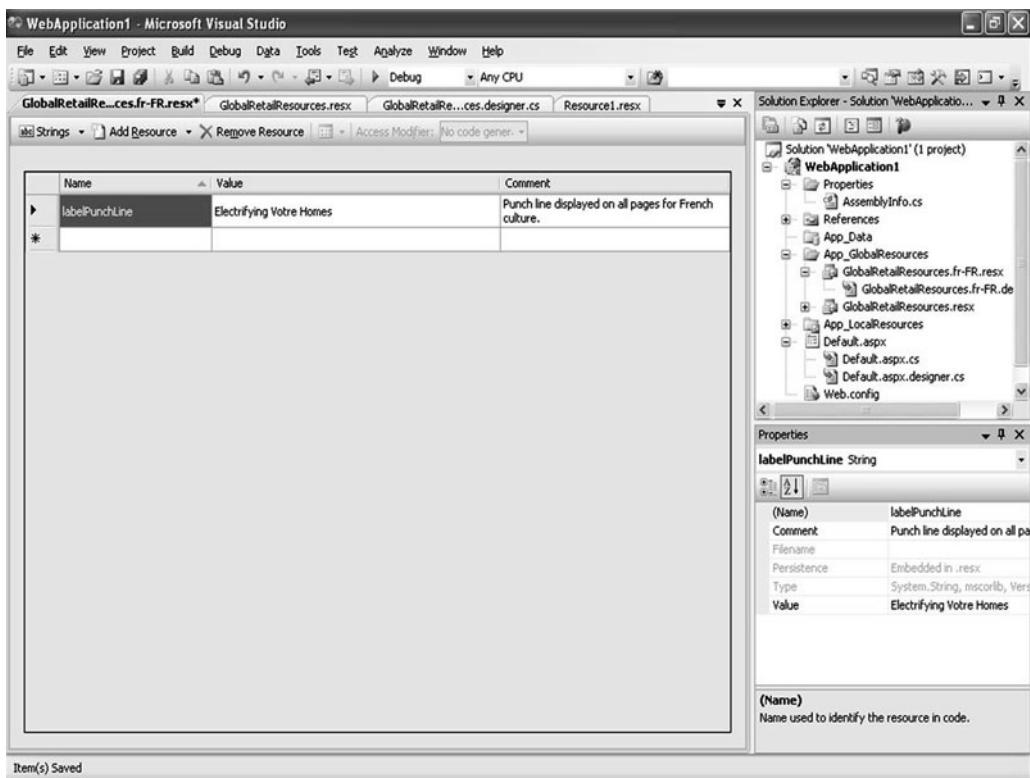
**TAKE NOTE ***

Note that after you finalize a base resource file, you can make a copy of it and rename the file as explained before.

Then, you can edit the values for the keys based on the other cultures as shown in Figure 8-6.

Figure 8-6

Editing values in the Resource Editor



However, the manual approach is used mostly for global resource files.

For local resource files, there is a simpler option, (i.e., the Generate Resource option).



USE GENERATE RESOURCE OPTION

To generate a local resource file for a page:

1. Open the page in the Design view in Visual Studio 2008.
2. Select Tools > Generate Local Resource from the main menu of Visual Studio 2008. The App_LocalResources folder will be created if it does not exist, and a base resource file will be created with the key-value pairs for all localizable properties of the page, and each control on the page that needs to be localized. The .NET Framework has added a Localizable attribute to those properties of controls that need to be localized. You can also apply this attribute, if you create any custom controls so that the control can be handled for localization properly.
3. Edit the values of the keys, and save the file.
4. Again, make a copy of the base file, rename it, and edit its key values for other cultures.

TAKE NOTE*

If you want to create a resource file outside of Visual Studio 2008, you can use the Resource Generator tool (Resgen.exe). For details on using the tool, you can refer to the Resource File Generator section on MSDN.

Localizing Web Applications Using Resource Files

Now that you have learned the basics of resource files and their creation, let us learn how you can finally use them to display localized content to your users.

In order to implement localization, you need to ensure that at runtime, the values of the localized elements of the page are read from the resource files. This can be done declaratively or programmatically.

Based on how this is done declaratively, localization using resources can be of two types: implicit and explicit.

TAKE NOTE*

It is not mandatory for the name of the control specified in the resource file keys to be the same as the name of the control in the code. It is the job of the **meta:ResourceKey** attribute to map these and retrieve and display the right localized value.

USING IMPLICIT LOCALIZATION

Implicit localization is applicable only for local resources; a **meta:resourceKey** attribute is added to all controls with one or more properties that need to be localized. So, for example, if you want to localize some properties, such as **Text**, **ToolTip**, **ForeColor**, and **BorderColor** properties, of the Label control named **lblTagLine**, then you must add the **meta:resourceKey** attribute as follows:

```
<asp:Label ID="lblTagLine" runat="server"
meta:ResourceKey="labelTagLine ">
</asp:Label>
```

In this code, when the **meta:resourceKey** attribute is added to any controls declaration, ASP.NET refers to the local resource file for the page and the current UI culture. It then uses the values of any properties of the control, which in this case is named **labelTagLine** in the local resource file.

Interestingly, the **meta:ResourceKey** attribute is automatically added to controls when you generate the local resource file by using the Generate Local Resource option in the Tools menu, as explained before. In fact, you can use the option repeatedly to update the base resource file for any new controls added. However, if a control is deleted from the page, its entries will not be removed from the resource file by using the option.

TAKE NOTE *

All properties of a control that are localizable and will be added to the local resource file by the Generate Local Resource option are marked with a small red icon to the right of their names in the Property sheet in Visual Studio 2008.

Implicit localization is a simple way to override values of localizable properties of a control. However, with implicit localization, the `meta:ResourceKey` attribute is added automatically to a control and applicable for all its localizable properties. In many cases, you may not need to use many or all of the keys added to the resource file. If you want to localize only one or two properties of the control, you can use ***explicit localization***.

USING EXPLICIT LOCALIZATION

Explicit localization is another declarative method for implementing localization in which expressions are assigned to property values of a control. The expressions explicitly define how the value of the property will be read from the resource file.

The syntax for using explicit localization in local resource files is:

```
<asp:Label ID="lblTagLine" runat="server"
Text="<%$ Resources: name_of_key_in_resource_file %>">
</asp:Label>
```

The syntax for using explicit localization in global resource files is:

```
<asp:Label ID="lblTagLine" runat="server"
Text="<%$ Resources: name_of_global_resource_file,
name_of_key_in_resource_file %>">
</asp:Label>
```

Note that in the explicit localization for global resource files, the name of the file is specified in the declaration.

Explicit localization is used more often in the case of global resources. For example, it is useful for text messages such as tagline, which are repeated in all pages although you do not want to maintain a separate local resource file for each page for this tagline's localization.

Declarative methods of localization do not allow you to dynamically retrieve resource values based on conditions. For this, you need to handle resource files programmatically.

To retrieve local resources programmatically, you can use the `GetLocalResourceObject()` method provided by the `HttpContext` and `TemplateControl` classes. The method accepts the name of the resource whose value needs to be retrieved from the local resource file of the page. The value returned by the method is an `Object` type and needs to be explicitly cast to the desired type. For example, you can retrieve the tagline text from the resource file as follows:

```
lblTagLine.Text = GetLocalResourceObject("labelTagLine.Text").
ToString();
```

In this context, you should understand the `Localize` control, which can hold static localized text. You can use this control as a placeholder for a large amount of localized text that needs to be set dynamically in the program. The `Localize` control is different from the `Literal` control in the following ways:

- It is visible in the Design view as well as Source view in Visual Studio, whereas the `Literal` control is not.
- Its value can't be directly edited, but the value of the `Localize` control can.

The `Localize` control can be used to show the static text as well as text entered by users, which might include malicious client scripts. This control is vulnerable to cross-site scripting (CSS) attacks.

RETRIEVING GLOBAL AND LOCAL RESOURCES

To retrieve global resources programmatically, you can use the `GetGlobalResourceObject()` method provided by the `HttpContext` and `TemplateControl` classes. The method accepts the class name and name of the resource. The class name of the resource is the base name of the global resource file. For example, if the tag line text is in a global resource file named

`GlobalRetailResources.resx`, then the `GetGlobalResourceObject()` method will be used as:

```
lblTagLine.Text = (String)GetGlobalResourceObject("GlobalRetailResources", "labelTagLine.Text");
```

TAKE NOTE *

You can also access resource files programmatically by using the `ResourceManager` class. For details on the class, refer to MSDN.

Another way to retrieve a global resource programmatically without the need for explicit type casting is by using strong typing. This method uses the following syntax:

```
Resources.<Class Name>.<Resource Name>
```

Resource is the name of a .NET Framework class of which all global base resources are a part. So, the tagline text defined in the global resource file `GlobalRetailResources.resx` can be retrieved as follows:

```
lblTagLine.Text = Resources.GlobalRetailResources.labelTagLine.Text
```

In addition to implementing globalization and localization as explained, you should also follow some best practices related to designing the application.

Utilizing Best Practices for Implementing Globalization

In .NET Framework 3.5, a new feature related to globalization and localization is the support for globalizing and localizing client scripts. Earlier, there was no way to access the resource files residing on the server in order to perform client side tasks, such as displaying data in a specific format or showing custom messages on client-side validation.

Some of the best practices that you can follow while implementing globalization and localization are:

- Separate the localizable content from the code completely.
- Avoid using images that have text because they become difficult to localize, and you will have to create a separate image for each supported culture.
- Ensure that the data is formatted according to the culture. Use the properties of the `CultureInfo` class to format the data.
- Avoid creation of text messages at runtime or by concatenating strings because the order of the strings forming a sentence is not always the same in all languages.
- Use the `dir` attribute in the `<body>` element to specify the direction in which the text is to be displayed—left to right or right to left. To do so, create a key by any name, such as `TextDir`, in all resource files of the application and set its value to LTR or RTL according to the culture. Then, use explicit localization to specify the value of the `dir` attribute in the `<body>` element of each page.
- Design the layout for flexibility by using tables. When using tables to contain controls in individual cells, remember the following points:
 - Do not define the height of the cell that contains a control that holds text, if it is not absolutely required. Any change in the amount of text due to size that is defined by the browser and due to localization may change the look of the UI.
 - Ensure that the `NoWrap` property is always disabled for tables so that word wrapping is enabled and text is not truncated when localized.
 - Let table alignment set according to the web browser instead of specifying them explicitly. This will prevent changes to the UI layout in different browsers after localization. Similarly, left and right alignments of cells may also affect the layout if the direction of text in a particular culture is different from what you perceived while designing.
- Design the layout vertically so that a horizontal scroll is not required if the text expands. In addition, keep enough space between controls so that their expansion does not cause overlap. For example, you can separate the label of a check box and the check box itself and leave appropriate space between them to incorporate expansion due to localization.

- Do not hard code the position and size of controls by specifying their absolute positions because the area occupied by the same text in different languages may vary. You should define the position and size of controls relative to the size of the browser. For example, you can define an element's height as one-third the height of the page.
- Specify the dimensions of tables in percentages instead of absolute values.
- Anticipate instances where resizing may be required, such as for dialog boxes that display a lot of text and may need resizing when the text is translated. Implement resizing solutions for such cases.
- Use variable-length character Unicode encoding for text as shown:

```
<globalization responseEncoding="utf-8" requestEncoding="utf-8"  
fileEncoding="utf-8" />
```

This is also compatible with ASCII and supports languages such as Arabic, in addition to the more common ones.

MORE INFORMATION

For more information on best practices for implementing Globalization, refer to MSDN.

Demonstrating Globalization and Localization

To demonstrate the globalization and localization concepts, let us develop a small application that uses the culture and resource file concepts.

This application contains a single web page that displays the following:

- A drop-down from which the user can select a language.
- A Thought of the day, “Honesty is the Best Policy,” which will change based on the selected language.
- The current date and time, displayed in the format supported by the selected culture.
- A button to view the next message. The caption of the button will also change as the language changes.



CREATE A GLOBAL APPLICATION

To develop this simple application based on globalization and localization:

1. Identify the culture to be supported by your application. Let us assume in this case that these are English, Spanish, Arabic, and French.
2. Now identify the way in which the culture settings need to be changed. As specified in the requirement, the settings need to be changed based on user input from a drop-down list.
3. Create a new ASP.NET web application project and design a form named Culture.aspx. The source code of the form is displayed here:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Culture.aspx.cs"  
Inherits="Culture" %>  
  
<!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>Thought of the Day</title>  
  </head>  
  <body>  
    <form id="form1" runat="server">  
      <div>  
        <div style="text-align: center">  
          <div style="text-align: left">  
            <table style="width: 600px; height: 155px">  
              <tr>  
                <td style="width: 62px">
```

```

        <asp:Label ID="Select" runat="server" Text="Select language"
Width="106px"></asp:Label></td>
        <td style="width: 100px">
            <asp:DropDownList ID="ddlLanguage" runat="server"
AutoPostBack="true">
                <asp:ListItem Value="-1">Select a language</asp:ListItem>
                <asp:ListItem Value="0">Arabic</asp:ListItem>
                <asp:ListItem Value="1">French</asp:ListItem>
                <asp:ListItem Value="2">English</asp:ListItem>
                <asp:ListItem Value="3">Spanish</asp:ListItem>

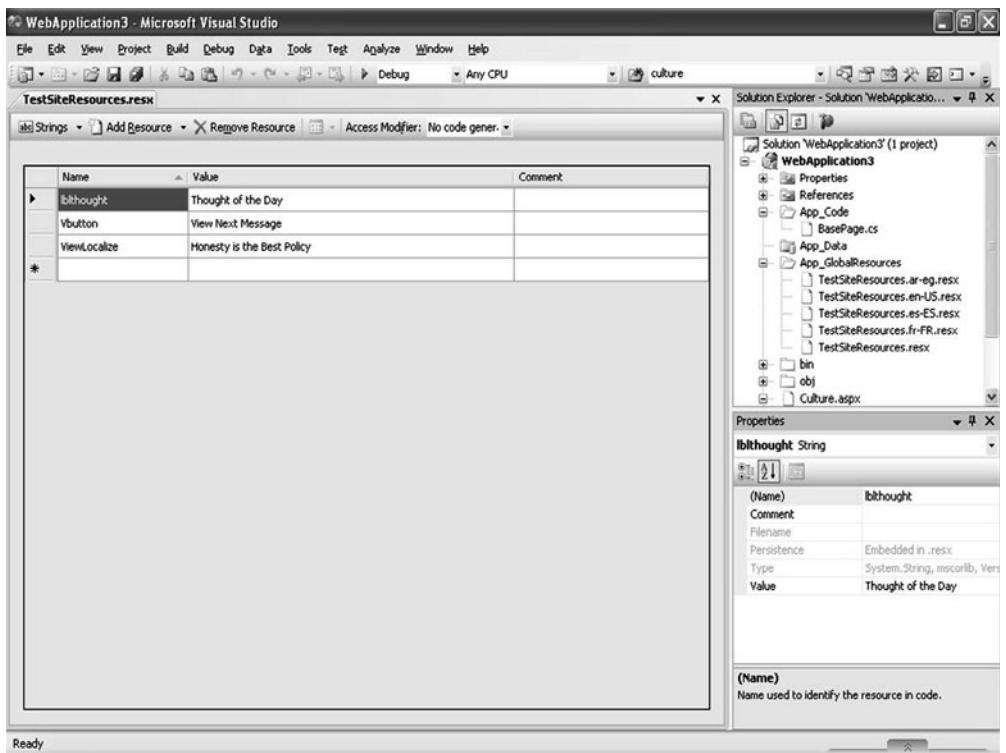
            </asp:DropDownList>
        </td>
        <td style="width: 100px">
        </td>
    </tr>
    <tr>
        <td style="width: 62px">
            <asp:Label ID="lblthought" runat="server" Text="" Width="162px"></asp:Label></td>
            <td>
                <asp:Localize ID="ViewLocalize" runat="server" Text="" Text="" ></asp:Localize>
                &nbsp;&nbsp;
            </td>
            <td style="width: 100px">
            </td>
        </tr>
        <tr>
            <td>
                Date format</td>
            <td>
                <asp:Label ID="Label1" runat="server" Text="No Date Selected"></asp:Label></td>
                <td style="width: 100px">
                </td>
            </tr>
            <tr>
                <td style="width: 62px">
                </td>
                <td style="width: 100px">
                    <asp:Button ID="Vbutton" runat="server" Text="" /></td>
                    <td style="width: 100px">
                    </td>
                </tr>
            </table>
        </div>
    </div>
</div>
</form>
</body>
</html>

```

4. Identify the data that needs to be formatted according to the selected culture and translated into the desired languages.
5. Create a base global resource file named TestSiteResources.resx to store the localizable content, as explained before. The file is shown in Figure 8-7.

Figure 8-7

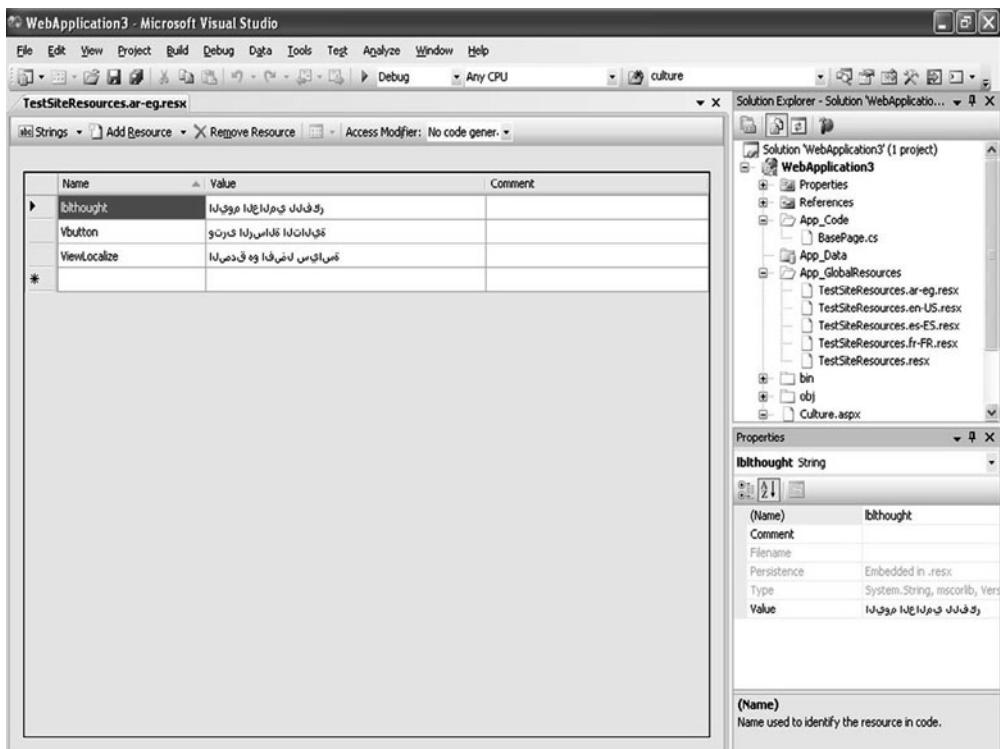
The global resource file



6. Right click the base file in the Solution Explorer, and select Copy from the shortcut menu.
7. Right click the App_GlobalResources folder, and select Paste to create a copy of the base file.
8. Rename the base file as TestSiteResources.en-US.resx, and update the field. Repeat the step to create the TestSiteResources.es-ES.resx, TestSiteResources.fr-FR.resx, and TestSiteResources.ar-EG.resx files. The TestSiteResources.ar-EG.resx file is shown in Figure 8-8.

Figure 8-8

The contents of the ar-eg base file



9. Now, edit the source code of the Culture.aspx file and implement explicit localization so that the controls on the form can retrieve data from the appropriate resource file. The following lines of code are updated to implement explicit localization as follows:

```
<asp:Label ID="lblthought" runat="server" Text="<%$ Resources:TestSiteResources, lblthought %>">
<asp:Localize ID="ViewLocalize" runat="server" Text="<%$ Resources:TestSiteResources, ViewLocalize %>"></asp:Localize>
<asp:Button ID="Vbutton" runat="server" Text="<%$ Resources:TestSiteResources, Vbutton %>" /></td>
```

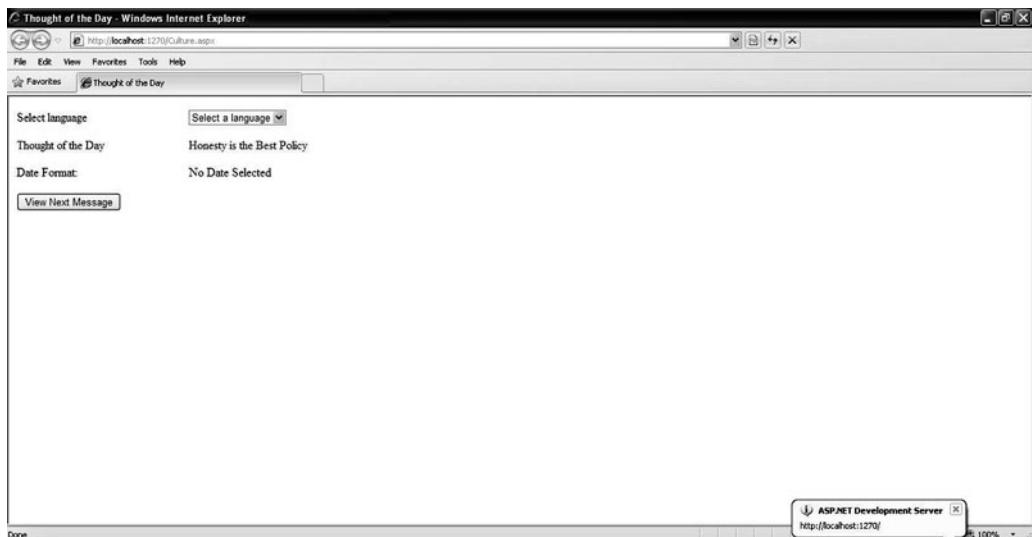
10. In the code-behind file, implement the date formatting according to the current culture as follows:

```
string selectedValue = ddlLanguage.SelectedValue.ToString();
if (selectedValue != "-1")
{
    switch (selectedValue)
    {
        case "0":
            System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("ar-eg");
            Label1.Text = DateTime.Today.ToString();
            break;
        case "1":
            System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("fr-FR");
            Label1.Text = DateTime.Today.ToString();
            break;
        case "2":
            System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("en-US");
            Label1.Text = DateTime.Today.ToString();
            break;
        case "3":
            System.Threading.Thread.CurrentThread.CurrentCulture = new System.Globalization.CultureInfo("es-ES");
            Label1.Text = DateTime.Today.ToString();
            break;
        default: break;
    }
}
```

11. Run the application. The output is shown in Figure 8-9.

Figure 8-9

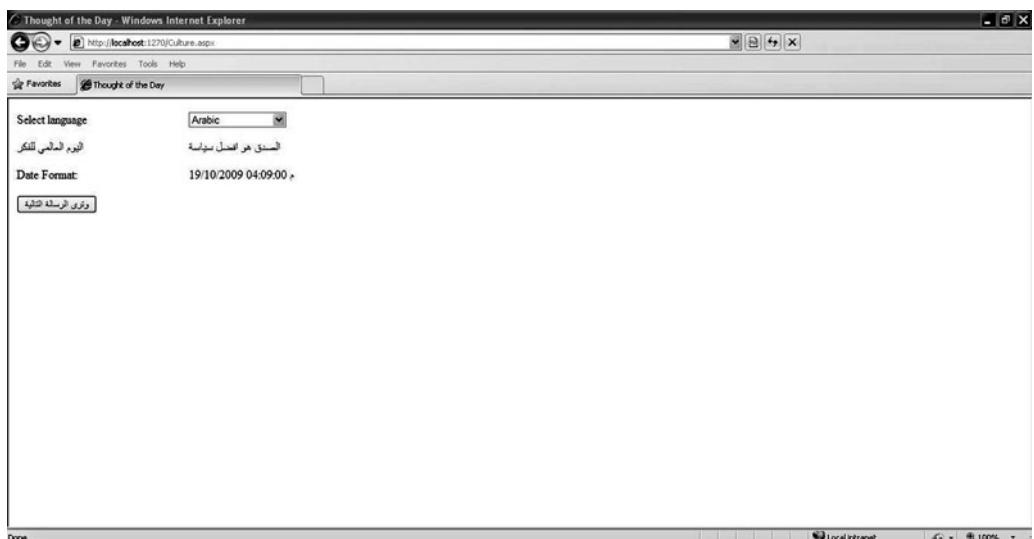
The output



12. On selecting Arabic from the list, the output obtained is shown in Figure 8-10.

Figure 8-10

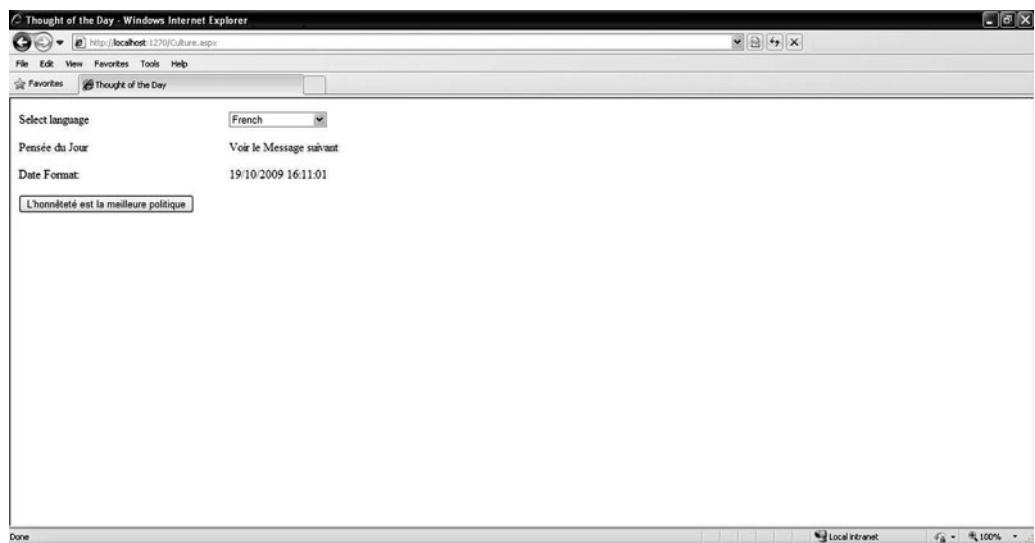
Selecting Arabic



13. On selecting French from the list, the output obtained is shown in Figure 8-11.

Figure 8-11

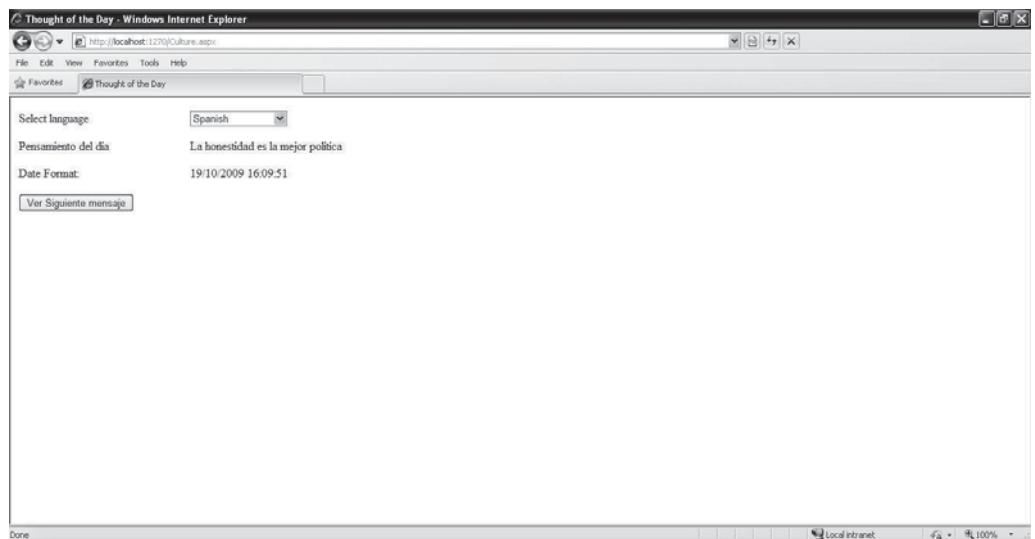
Selecting French



14. On selecting Spanish from the list, the output obtained is shown in Figure 8-12.

Figure 8-12

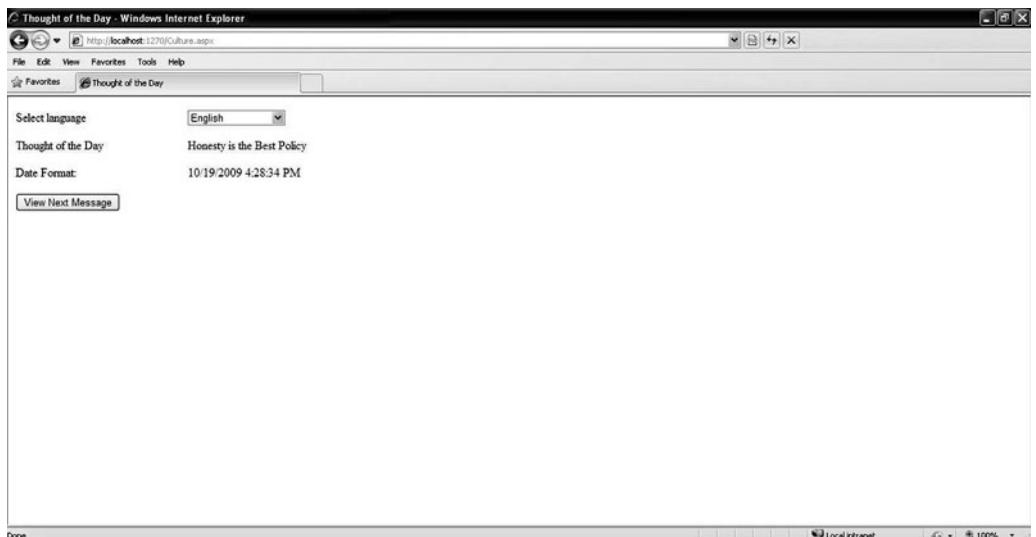
Selecting Spanish



15. On selecting English from the list, the output obtained is shown in Figure 8-13.

Figure 8-13

Selecting English



TAKE NOTE *

The steps for testing may vary with the browsers. There are plug-ins for other browsers that make the testing process easier by just adding a button in the corner. You can press this button to toggle cultures.

■ Configuring Accessibility



THE BOTTOM LINE

Considering the ever-growing popularity and usage of the Internet for varying purposes, its user base is increasing. A significant part of this user base is people who either have some disability or special needs. As a result, they need to use the Internet-based application differently than a normal user. For example, a user may have a limb or vision disability and be using a special input device for the computer, such as a text-Braille device. Another example is a color-blind person who cannot tell the difference between available and unavailable seats in a cinema's ticket-booking Web site because they have problems with colors such as yellow on green or brown and purple or red on blue. For such users, web applications need to be made more accessible. A web application is accessible when it is used with equal ease by users with special needs or disabilities. Even if your Web site does not require accessibility, it is still a good practice to design sites in such a way that accessibility options may be added at a later time.

The importance of accessibility can be gauged from the fact that many countries have formed guidelines and even legislation pertaining the inclusion of and adherence to accessibility features in Web sites, such as Section 508 of the Rehabilitation Act of the United States and PAS78 of the United Kingdom. The World Wide Web Consortium (W3C), an organization that has had a history with web-related standards, has made a commendable move in this direction. The Web Content Accessibility Guidelines (WCAG) were published in 1999 and are increasingly being used as the standard for developing accessible web applications. The second version was published in 2008.

Understanding Accessibility Needs in a Web Application

The accessibility needs of a web application can be divided into three categories: data input, data output, and site navigation.

Data input refers to support for different types of data input devices to ensure that users with special abilities are able to submit their input. For example, a UI should not be designed so that it only accepts input from a pointing device. Users with mobility-related disabilities may not be able to use a pointing device effectively. Therefore, for every operation performed through a pointing device there should be an alternative, such as keyboard commands.

Data output refers to support for different types of data output devices and formats to ensure that users with special abilities are able to comprehend the output. For example, the screen output should use a format (such as audio or Braille) that screen reader software is able to interpret and render for a user with a disability.

Site navigation refers to designing the layout and content of the site so that users do not lose their way and can easily access all pages while trying to browse through the site. For example, disabling the link of the web page that a user is currently viewing will prevent them from being mislead and clicking on the same link again.

Input and output technologies that simplify the task of entering data and comprehending the output of a computer application are called assistive technologies.

SUPPORTING ACCESSIBILITY IN ASP.NET CONTROLS

Considering the importance of accessibility needs and guidelines, ASP.NET provides features to design and develop accessible Web sites. Let's identify these features.

Using Textual Alternatives for All Nontext Elements

Images in an ASP.NET page can have an alternative text specified so that if the image is not displayed, the associated text can be seen. This is possible in the following ways:

- In the Image, ImageMap, and ImageButton controls, you can set the `AlternateText` property to a descriptive alternate text for all images that have functional importance for the Web site. If an image is only an accessory and does not have any functional needs, you should set the value of the property to an empty string. You must set the `GenerateEmptyAlternateText` property to `true` for these controls if you want to set the `AlternateText` property to an empty string.
- Some other controls in ASP.NET, such as TreeView, Menu, and Web Parts automatically render alternate text for images used for their functioning, such as the images used for expand/collapse functionality in a TreeView control.
- Various other controls also allow you to specify the alternate text for images used. For example, the HyperLink control's `Text` property can be set to define an alternative text if the `ImageUrl` is set to the path of an image file.

Note that the `AutoPostBack` property needs to be `true` for all controls. As such, the control relies on the client script.

TAKE NOTE*

In the `Text` and `ErrorMessage` properties of validator controls, specify descriptive and meaningful messages. Using an asterisk (*) against controls in a web page denotes that the control is mandatory. However, screen readers cannot suggest this. Therefore, it is good practice to provide descriptive error messages, such as "You must enter your work phone number."

Using UI Elements That Can Be Interpreted by Assistive Technologies

These are the accessibility guidelines for UI elements:

- Set a caption for the `Caption` property of the Table control. The caption can be interpreted by the screen reader to convey the purpose of the table for users with a disability. In addition, navigation within a table can be simplified by implementing a way to identify column headers and footers, and associating cells with them. To include headers and footers, you can use the `TableHeaderRow` and `TableFooterRow` classes and set the `TableSection` property to `TableHeader` or `TableFooter` values, respectively. This

will allow screen readers to interpret the headers and footers in the table. Also, if the table spans multiple pages, by using the `TableHeaderRow` object, you can repeat table headings on each.

- Add content to the `TableHeaderRow` object by using the `TableCell` control. You can associate each cell in the table with its heading by setting the `AssociatedHeaderCellID` property of the `TableCell` object.
- Set the `Caption` and `UseAccessibleHeader` properties for other controls that render HTML tables—such as `Calendar`, `DetailsView`, and `GridView`—to ensure that they have the information that's needed for interpretation by assistive technologies. When the `UseAccessibleHeader` property is set to `true`, a `scope` attribute is added to each header cell. The `scope` attribute can be set to a column header or a row header thereby indicating how it is related to a cell. You will see an example of how `GridView` is made accessible later in the section.
- Set the value of the `AssociatedControlID` property of the `Label` control to the ID of the control to associate the `Label` control with the control.
- Set the value of the `Text` property to different values for different instances of the same controls, like `LinkButton`, to identify their purpose so that the screen readers can distinguish between each control instance on the form.
- Describe the purpose of `LoginName` for the screen readers by setting its `ToolTip` property.

Ensuring User Agent Independence

An application should not be dependent on the user agent, such as the web browser, being used to access the application. You can ensure this by:

- Using multiple style sheets so that the user can control the color scheme of the UI and the size of the text.
- Including `<noframes>` and `<noobject>` tags for agents that do not support frames and objects.
- Avoiding client script for functionally important tasks or writing accessible client script that is not dependent on I/O devices to ensure that all content is available to assistive technologies.

ASP.NET provides the `HtmlLink` control that allows you to easily specify one or more style sheets.

You can avoid the need for client script when using controls that use the `LinkButton` control by creating templates and adding `Button` controls for the functions that the `LinkButton` control is expected to perform.

TAKE NOTE *

Built-in validator controls, except for `CustomValidator`, perform validation on the server side. However, by default, the controls also render client script to perform validation on the client side. This redundancy can be eliminated by setting the `EnableClientScript` property of the control to `false`. In that case, the use of client script will be disabled and the validation check will now only be performed on the server side.

Ensuring Ease of Navigation

Supporting keyboard operations is a very important accessibility feature that helps users navigate easily. Many users may not have access to or may not be able to use pointing devices. To simplify navigation for such users, you should utilize keyboard-related settings. In ASP.NET, you can support keyboard operations by:

- **Setting the `DefaultFocus` property of an input control on the web page:** This property will by default bring the focus to the control when the page is loaded so that the user can start typing the input without the need to explicitly use a pointing device and shift the cursor to the control.
- **Shifting focus to a control that needs attention by using the `SetFocus()` method:** For example, on form validation, if a few fields have not been filled in correctly, then

you can use `SetFocus()` to shift the focus to the first of these fields. It is not possible for all controls to receive focus. Only some of the commonly used controls can receive focus.

- **Defining the tab order for navigation on the page by using the Tab key:** This can be done by setting the `TabIndex` property of controls. The Tab key will shift focus among the controls based on the order of the `TabIndex` values defined for the controls.

TAKE NOTE*

For things such as registration forms, controls are sometimes laid out vertically in columns, but if you put the controls in a table, the tab order will go left to right. So the tab order should follow a logical progression rather than the location.

- **Defining access keys for controls such as buttons and menus on a page by setting the AccessKey property of the controls to a letter or number:** The user will be able to use the control without a pointing device by pressing the Alt+defined access key combination on the keyboard. For a `TextBox`, you should set the `AccessKey` property of the `Label` control associated with the `TextBox` control through `AssociatedControlID` property.

Other accessibility features that ensure ease of navigation are:

- **Setting the DefaultButton property:** Setting the `DefaultButton` property of a button to a `Button` or `ImageButton` control's ID in a container element such as a `panel` or `page` will emulate the action of the button click. Again, the need for a pointing device will be eliminated. Note that this is only applicable in an `asp:Panel` control.
- **Skipping repetitive links:** To simplify navigation by assistive technologies, you need to ensure that the devices do not have to read repetitive links on pages, such as sets of links repeated on all pages of a Web site, for navigation. Some controls in ASP.NET, such as `Menu`, `TreeView`, `SiteMapPath`, `CreateUserWizard`, and `Wizard`, support a `SkipLinkText` property that can be set to a textual value to skip the links.

Another guideline to follow: You must never assume a maximum width for a background. Flexible page layouts allow exact scaling of the page content when the text size of the page in the web browser is increased. These layouts are also helpful to users who have specialized accessibility settings. Divide a page's layout into sections by using `Panel` controls. You can then describe controls in each panel by setting the `GroupingText` property of the `Panel` controls.

+ MORE INFORMATION

To develop an accessible Web site, it is important that you are aware of the general accessibility guidelines for designing the Web site. You can refer to the Accessibility Support in ASP.NET section on MSDN and go to <http://www.w3.org> for more information.

TESTING ACCESSIBILITY

Visual Studio facilitates testing web pages or entire web applications for compliance against WCAG and Section 508 standards. You can also automate the process of testing for accessibility.



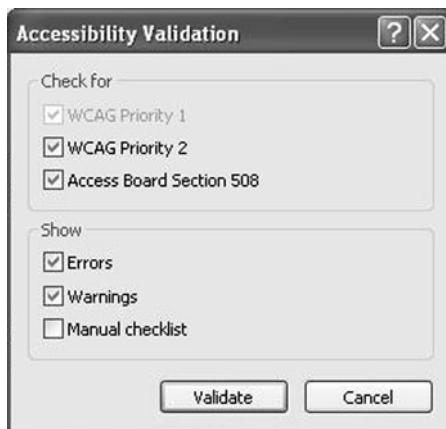
TEST ACCESSIBILITY OF A WEB APPLICATION

To test a web page for compliance:

1. Open the web page in Visual Studio 2008.
2. Select Tools > Check Accessibility from the main menu. The Accessibility Validation dialog box appears as shown in Figure 8-14.

Figure 8-14

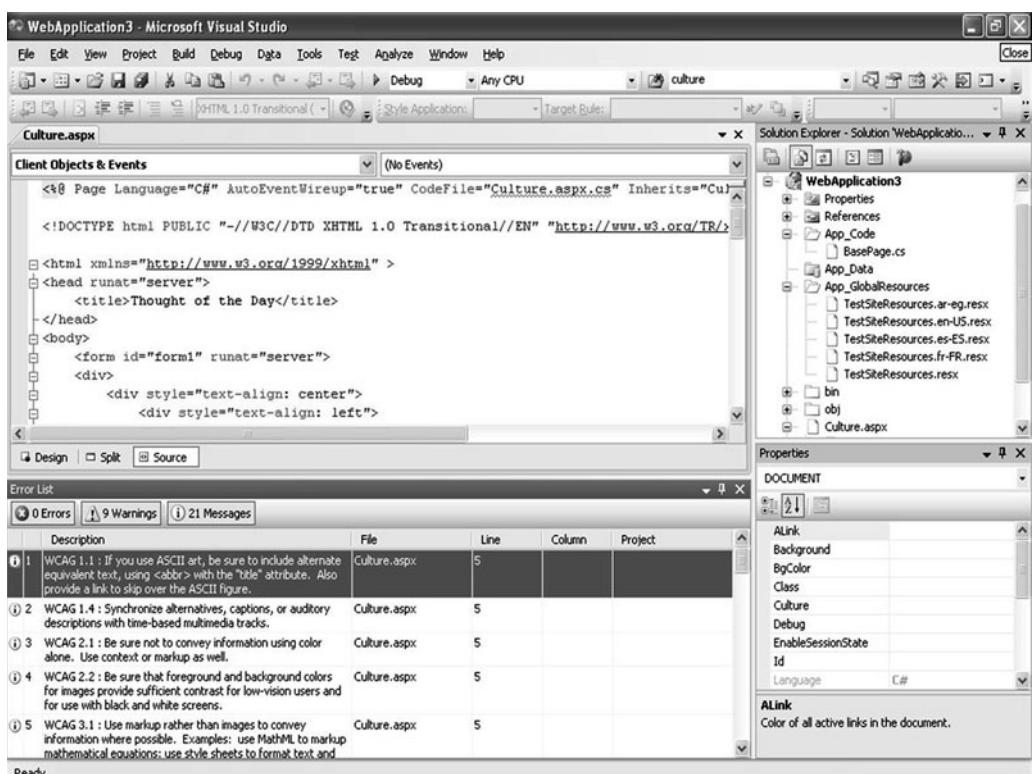
Accessibility Validation dialog box

**Figure 8-15**

Error list window

TAKE NOTE *

There are also tools online to put in a site's URL to check accessibility problems. This is often helpful if your site is already up or if your site uses a content management system so you don't have full control over content.

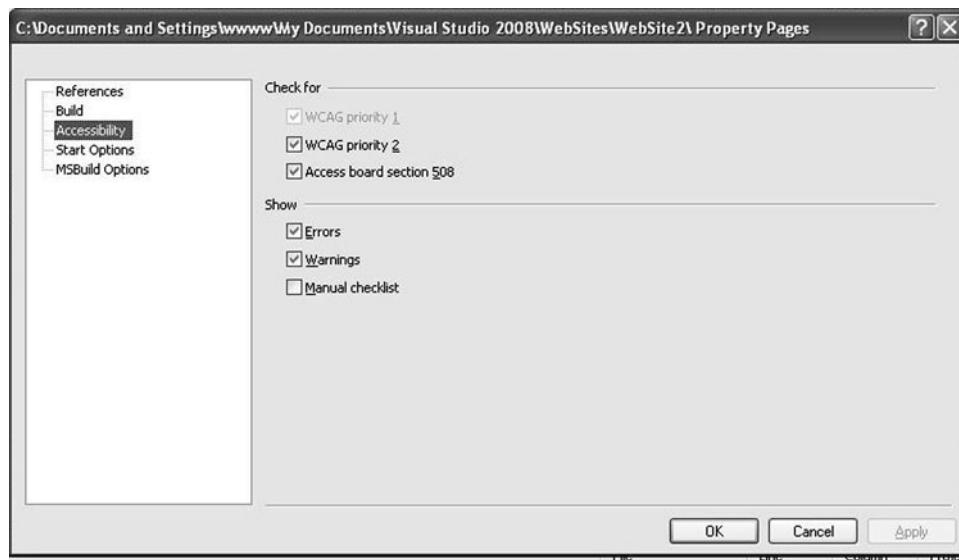
**AUTOMATE TESTING FOR ACCESSIBILITY**

To automate accessibility validation:

- In the Solution Explorer, right click the Web site name, and select Property Pages from the shortcut menu. The Property Pages dialog box appears for the Web site.
- In the left panel of the dialog box, select Accessibility as shown in Figure 8-16.

Figure 8-16

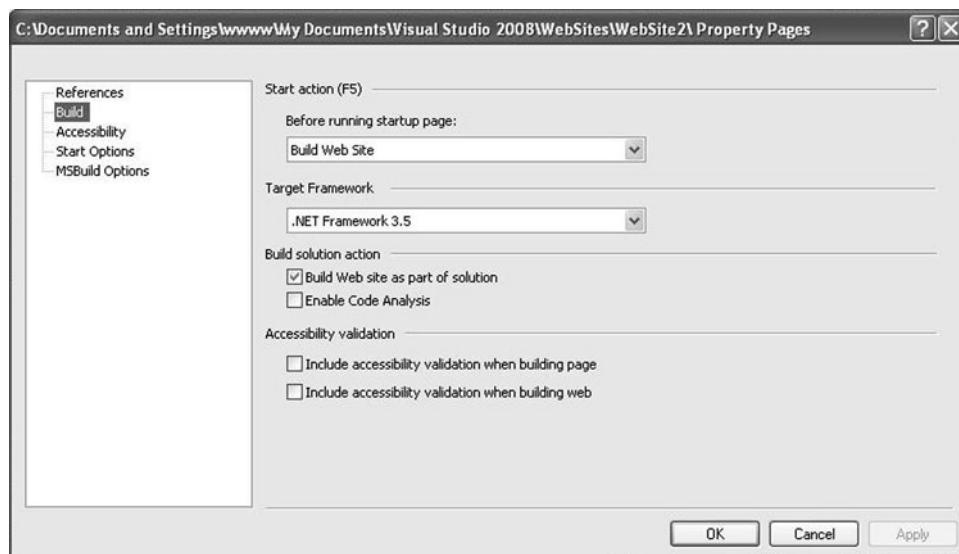
Selecting accessibility in Property Pages



3. Under the Check for section, select the desired guidelines to be checked against. Under the Show section, select the level of messages that you want to view after accessibility validation.
4. Click Apply.
5. In the left panel of the dialog box, select Build as shown in Figure 8-17.

Figure 8-17

Selecting build options



6. Under the Accessibility Validation section, select one or both of the check boxes. Select the Include accessibility validation when building page to automate the accessibility check for a single page. Select the Include accessibility validation when building web to check the entire Web site.
7. Click OK

CERTIFICATION READY?
Implement globalization
and accessibility.

7.3



USE THE CULTUREINFO CLASS

To set the language to Spanish in the Mexican culture in an application:

Use the `CultureInfo` class for the `Date` class. Use the following code:

```
DateTime dt = DateTime.Now;
CultureInfo ci = new CultureInfo("es-MX");
string newCurrTime = dt.ToString("d", ci);
```

SKILL SUMMARY

In this lesson, you learned about localization and globalization. You learned how to translate web pages using the resource files. Local resources are stored in the App_LocalResource folder, while global resources are stored in the App_GlobalResource folder. You also learned how to generate the local resource using Visual Studio. In addition, you learned to use the `<%$ Resource:ClassKey, ResourceKey %>` syntax to set the Text property using the resource value. You learned about various best practices for globalization in this context.

This lesson described setting culture-specific information using the Culture and UICulture properties and the CultureInfo class. You learned how to create the language-specific local resources and global resources. You also learned that the Localize control can be used to display the localized static text on the web page.

Finally, you learned about accessibility, how ASP.NET controls support it, and various guidelines for the ASP.NET controls support for accessibility. You also learned how to improve visual and keyboard accessibility in web applications. Then, you learned how to test accessibility with the help of Visual Studio.

For the certification examination:

- Implement globalization and localization techniques and using HTML layout best practices.
- Implement the public accessibility guidelines to configure and improve accessibility of forms that accept user input.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. The _____ resources of ASP.NET provide support for multiple languages in your web applications.
2. All local resources should be stored in the _____ subfolder.
3. _____ resources can be accessed from any page in a Web site.
4. When configuring culture settings, after setting the Culture and UICulture properties, you need to call the base _____ method of the page.

True / False

Circle T if the statement is true or F if the statement is false.

- | | | |
|---|---|--|
| T | F | 1. Localization allows your application to be used by people in different parts of the world who speak different languages and follow different formatting standards for numbers, currency, and dates. |
| T | F | 2. Local resource files are named using the format <PageName>[.language].res. |
| T | F | 3. When implementing globalization, use absolute positioning and sizes for controls. |
| T | F | 4. Implicit localization is applicable only for local resources. |
| T | F | 5. Use the <code>String.Compare</code> method when you want to compare or sort string values. |

Multiple Choice

Circle the letter that corresponds to the best answer.

1. Which class is used to define culture-related information?
 - a. Culture.Information
 - b. Culture.Info
 - c. CultureInfo
 - d. CultureInformation

2. Which of the following should you use while implementing globalization for an application?
 - a. Implement a horizontal layout to avoid vertical scroll.
 - b. Define specific dimensions for cells.
 - c. Implement creation of text messages at runtime.
 - d. Specify the dimension of tables in percentages.

3. To which subfolder should you add your files to create global resources?
 - a. App_GlobalResource
 - b. App_Resources
 - c. App_LocalResources
 - d. App_GlobalResources

Review Questions

1. How can you retrieve global resources from a page programmatically?
2. List some guidelines that you need to follow to improve visual accessibility.

■ Case Scenario

Scenario 8-1: Displaying Information in Multiple Languages

You are developing a web application to capture survey information. The web application accepts various user inputs. You need to display some static information in multiple languages. How will you display the information?



Workplace Ready

Using Localization and Globalization

You are developing a Web site for traffic awareness. You need to display various traffic rules and regulations in various languages for different countries. How will you achieve this goal?

Enhancing Web Application Security and Performance

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Using caching to improve performance.	Implement session state, view state, control state, cookies, cache, or application state.	7.5
Securing web applications.	Configure providers.	1.1
Securing web applications.	Configure authentication, authorization, and impersonation.	1.2

KEY TERMS

application data caching
authentication
authorization
fragment caching

impersonation
membership provider
output caching

While developing a web application, you implement all the functional requirements of the application. Nonfunctional requirements are also crucial for any application to ensure a good user experience. In the case of web applications, particularly; two nonfunctional requirements that are imperative include performance and security.

A web application is accessed over the Internet, and users of web applications are often impatient when waiting for pages to be displayed. They often close pages that take too long to display for various reasons, such as assuming that the network bandwidth is down or that the server is not responding. Therefore, if you want users to visit and browse through a web application, you should create a fast-responding application. The performance requirement is crucial for web applications.

Similarly, considering that data is exchanged with the web application over a network, it is prone to misuse by malicious users who eavesdrop on the network's communication channels. For example, crucial data of banking organizations that provide online access is at stake when accessed over the Internet by customers. Securing this type of web application is a must. In this lesson, you will learn about the performance and security features supported by ASP.NET.

■ Using Caching to Improve Performance



Caching refers to storing frequently used data in memory to serve repeated requests of the same or a subset of the data. The cached data can be stored in the memory of any machine that is used to access the application. These can include the client that sends a request, any proxy servers that route the request to the main server, and the server itself. Caching, when used correctly, serves as a technique for improving the performance of an application because it is faster to retrieve data from the memory than from a database stored on a secondary storage device.

To understand the need for caching, consider the case of an online shopping Web site where a specific page in the catalog is often viewed by users. This page can be cached in memory instead of all the product data on the page being retrieved from the database. As a result, the catalog will be displayed much faster to the user.

Cached data can be stored for an indefinite period or a specific period. The data in a cache expires if it has changed at the back end and needs to be updated. Say, the price of a product in the catalog has changed. In that case, you cannot display the cached version of the catalog page from the memory. You will have to retrieve the page again from the server with the updated data.

If data is cached on the client machine, then the user may explicitly clear the browser cache if desired.

ASP.NET supports three types of caching:

- Output caching
- Fragment caching
- Application data caching

Understanding Output Caching

Output caching refers to caching the output of an entire page requested by users. It is useful for pages that are requested frequently by multiple users of the Web site, such as a Login page of a Web site or a Search page of a search engine. Once a page is cached, it can be rendered by retrieving it from the cache instead of processing the entire code for the page again on the server to render it.

To use output caching, you need to first make a page cacheable and set its cacheability-related properties. You make a page cacheable declaratively or programmatically.

IMPLEMENTING OUTPUT CACHING DECLARATIVELY

Declaratively, you can include the @OutputCache directive in the page and set its attributes. Some of the main attributes are:

- **Duration:** A mandatory attribute that defines the time (in seconds) after which the cached page will expire. After the specified duration has passed, the page will need to be retrieved again from the server and be recached.
- **VaryByParam/VaryByControl:** An attribute that can be set to a list of keynames from the query string of the page. It enables multiple versions of the page to be cached based on the values of the specified parameters in the list. For example, if the URL of a page showing a description of a product is ~\catalog.aspx?productID=78, then you can cache the description pages of different products by setting the value of this attribute to ProductID. If you want to vary versions of pages based on more than one key from the query string, you can use a semicolon-separated list. To vary the pages by all parameters, set the value of the attribute to an asterisk(*). You can also vary versions of pages based on control value submitted by the POST method using the VaryByControl

attribute. You also need to use either the `VaryByParam` or `VaryByControl` attribute mandatorily. Other attributes for varying versions of cached pages include `VaryByHeader` and `VaryByCustom`.

- **Location:** An optional attribute that you can set to specify one or more of the locations where caching can occur, such as client, server, or any proxy server on the way. The value of this property can be a member of the `OutputCacheLocation` enumeration, which includes Any, Client, Downstream, Server, None, or ServerAndClient. Note that setting the property to None turns off output caching.
- **NoStore:** An optional attribute that you can set to indicate whether the cached pages can be stored on a secondary storage device. This is useful if the pages contain crucial or sensitive information.

The following code example sets some of the caching-related attributes of a page having the following URL `~/default.aspx?name=BenJones&ID=0988`:

```
<%@ OutputCache Duration="90" VaryByParam="name;ID"
Location="ServerAndClient" %>
```

In the preceding code, the `default.aspx` page will be cached for 90 seconds before it expires on the client and the server. Multiple versions of the page will be cached based on the `name` and `ID` parameters.

MORE INFORMATION

For information on the other attributes, refer to MSDN.

IMPLEMENTING OUTPUT CACHING PROGRAMMATICALLY

You can also perform the tasks achieved through the preceding attributes programmatically by using the members of the `System.Web.HttpCachePolicy` class. Properties of the class include `VaryByParams` (same purpose as the attribute), `VaryByHeaders`, and `VaryByContentEncodings`. These are used for caching multiple versions of the same page based on different criteria.

More settings related to caching can be implemented programmatically using the methods of the `HttpCachePolicy` class. Some of the commonly used methods are:

- **SetExpires()**: Sets a date and time at which a cached page expires. The date and time is passed as a `DateTime` type argument to this method. The date and time set using this method is absolute.
- **SetCacheability()**: Describes where caching can occur (corresponding to the `Location` attribute). The desired cacheability is passed as an argument to this over-loaded method as a member of the `System.Web.HttpCacheability` enumeration. When setting expiration through the `SetExpires()` method, you will also need to set the cacheability to `Public` by invoking the `SetCacheability()` method.
- **SetValidUntilExpires()**: Defines whether the page is expired due to cache invalidation headers, even when its expiration date and time have not been reached. If you pass `true` as its argument, the invalidation headers will be ignored and the page will be valid until it expires. Note that this method is invoked with `true` as the argument when you include the `@OutputCache` directive in your page.
- **SetSlidingExpiration()**: Defines whether sliding expiration is enabled. Sliding expiration causes the cached page to expire if the user does not reload the page within the duration specified by the `SetExpires()` method. You can pass a value of `true` to enable sliding expiration and `false` to use absolute expiration.

TAKE NOTE *

For a complete list of methods of the `HttpCachePolicy` class, refer to MSDN.

The following code example sets the caching-related properties programmatically on a page with the following URL `~/default.aspx?name=BenJones&ID=0988`:

```
Response.Cache.SetExpires(DateTime.Now.AddSeconds(90));
Response.Cache.SetCacheability(HttpCacheability.Public);
Response.Cache.VaryByParams["name"] = true;
Response.Cache.VaryByParams["ID"] = true;
```

In the preceding code, an instance of the `HttpCachePolicy` class is obtained by using the `Cache` property of the `Response` object for the page. It is then used to access properties and

invoke methods of the class. The expiration is set to 90 seconds by using the `SetExpires()` method. In addition, cacheability is set to `Public` (for client and server). Next, multiple version caching is implemented by setting the `VaryByParams` property. The versions will vary based on the parameters contained with the `[]`.

PERFORMING ADVANCED OUTPUT CACHING TASKS

In addition to setting the caching-related attributes to enable a page to be cached, you can also perform some advanced output caching tasks such as:

- Invalidating a cached page explicitly
- Implementing file and cache item dependencies to remove a page from cache
- Configuring caching
- Using cache substitution

INVALIDATING CACHED PAGES

Say, you are developing an online news portal, which updates the news articles displayed every 2 minutes. However, users are also allowed to add their comments for every news article on the portal. You have enabled caching for the news article pages as follows:

```
<%@ OutputCache Duration="120" VaryByParam="articleID" %>
```

However, you want to explicitly invalidate a news article page if a user has posted a comment within the 2 minutes for which the cached version of the page is used, so that the most recent version of the page is displayed to users. For this, you will need to write code to invalidate the cached page. To do so, when a user submits a comment on the article, you can submit a Boolean variable `updatedMessageBoard` in the query string or a hidden field with its value set as `true`.

The value of this variable can then be checked and invalidation code can be written to invalidate the page. To do so:

1. Define an event handler for the `ValidateCacheOutput` event as follows:

```
public static void ValidateCache (HttpContext context, Object
data, ref HttpValidationStatus status)
{
    ...
}
```

In the preceding definition, the last argument is a member of the `HttpValidationStatus` enumeration, which has the values `Invalid`, `IgnoreThisRequest`, and `Valid`.

2. Add code to check the value of the `updatedMessageBoard` variable and set the validation status accordingly as shown in the following code snippet:

```
if (context.Request.QueryString["updatedMessageBoard"] == "false")
    status = HttpValidationStatus.Invalid;
else if
(context.Request.QueryString["updatedMessageBoard "] == "ignore")
status = HttpValidationStatus.IgnoreThisRequest;
else
    status = HttpValidationStatus.Valid;
```

TAKE NOTE*

Assume that the code to set the value of the `updatedMessageBoard` variable has been set when a user submits a comment for an article. In addition, the variable value should be set to `false` after the cached page has been invalidated, so that it can be set to `true` again when a user enters a comment again.

3. Invoke the `AddValidationCallback()` method to invoke the preceding handler. The following code shows how the callback method will be invoked in the `Page_Load` event handler:

```
Response.Cache.AddValidationCallback(new  
HttpCacheValidateHandler(ValidateCache), null);  
if(!IsPostBack)  
{  
    ...  
}
```

MORE INFORMATION

For information on each member, refer to MSDN.

The method accepts a `System.Web.HttpCacheValidateHandler` delegate.

IMPLEMENTING FILE AND CACHE KEY DEPENDENCIES

Apart from using custom criteria to invalidate a cached page, you can specifically remove pages from a cache based on the following two criteria:

- Changes made to data store file or any other files in the back end
- Changes made to the cache

To implement the dependency of a cached page on a file, after you have enabled caching for the page and set the associated attributes, you can use the `AddFileDependency()` method of the `HttpResponse` class. This method accepts a string value defining the full name of the file on which a cached page is dependent. For example, consider the online newspaper portal example. If caching the news article web pages is dependent on a text file containing the text for the articles, then you can add the file dependency as follows:

```
// @ OutputCache directive comes here  
...  
// Code to save the article ID in a String variable id  
String articleFile = "~/"+ id + ".txt";  
Response.AddFileDependency(articleFile);
```

In the preceding code, a file name formed dynamically based on the article ID is passed as an argument to the `AddFileDependency()` method, which adds the file name to the collection of file names on which the news article page is dependent. When the content of the file changes, the cached page is removed from the cache so that the fresh page can be generated on the server and sent to the client.

TAKE NOTE*

Assume that the code to set the value of the `updatedMessageBoard` variable has been set when a user submits a comment for an article. In addition, the variable value should be set to false after the cached page has been invalidated, so that it can be set to `true` again when a user enters a comment.

TAKE NOTE*

The dependency methods discussed here are not applicable for invocation from a `.ascx` file.

You can also define dependency on more than one file by using the overloaded `AddFileDependencies()` method. One overloaded form of the method accepts an array of strings containing the file names, and the other overload accepts an `ArrayList` instance.

Similarly, you can make a page dependent on an item in the cache so that the page is removed when the item is removed from the cache. For this, the `HttpResponse` class provides the `AddCacheItemDependency()` method. The method accepts the name of the key in the cache as a string. You will learn more about the `Cache` class that represents a cache later in the lesson.

Regardless of any dependencies required, you can explicitly remove an item from the cache by invoking the `RemoveOutputCacheItem()` method.

You can also configure dependency of a cached page on SQL Server database objects. This is crucial for applications that pick up data from a SQL Server database. For this, you can use

the `SqlDependency` attribute and set it to the name of the database object on which the page is dependent as shown:

```
<%@ OutputCache Duration="60"
SqlDependency="MyDB:MyTable" VaryByParam="none" %>
```

MORE INFORMATION

For details about the `RemoveOutputCacheItem()` method and the `SqlDependency` attribute, refer to MSDN.

CONFIGURING OUTPUT CACHING

You have learned about output caching for a single page. You can also set up output caching for an entire application by making configuration settings in the `web.config` file of your application.

Here are some useful settings that you can make in the `web.config` file for output caching:

- **Enable or disable caching:** Use the `enableOutputCache` attribute in the `OutputCache` section in `web.config` to enable or disable caching as follows:

```
<system.web>
  ...
    <caching>
      ...
        <outputCache enableOutputCache="true|false">
        </outputCache>
      ...
    </caching>
  ...
</system.web>
```

- **Configure cache profiles:** Define output caching-related settings that are applicable to pages in a web application by creating a cache profile. For example, you can create a cache profile for all pages that you want to cache on the client using the following code:

```
...
<caching>
<outputCacheSettings>
  <outputCacheProfiles>
    <add name = "MyClientCacheProfile" enabled =
"true" duration = "120" location = "Client" varyByParam =
"productID" noStore = "false"/>
  </outputCacheProfiles>
</outputCacheSettings>
</caching>
...
```

As shown, the `outputCacheProfile` element uses its child element to add a profile. You can add multiple profiles in the same manner. You can also remove a profile or use the `clear` child element to remove all profiles.

The settings made in the profile can be applied to one or more pages by using the `CacheProfile` attribute in the `@OutputCache` directive. The name of the profile defined in the configuration file is specified as the value of the attribute in the directive.

You can have multiple cache profiles for different pages of an application. This allows you to configure a separate set of cache-related settings for each page and easily update or modify those pages individually. For example, different pages may need to be cached for different durations based on the nature of the dynamic content that they contain. In addition, some pages may need to be expired on absolute expiration whereas others may

MORE INFORMATION

For detailed information on the `OutputCache` element and the parent element, refer to MSDN.

require sliding expiration, and so on. Therefore, to simplify configuration in such cases, ASP.NET gives you an option to create multiple profiles.

USING CACHE SUBSTITUTION

Consider the example of the online shopping Web site catalog page that can be cached because of the high number of hits. Suppose this page contains the instance number of your visit to that page. This means that it displays the number of times that you have visited this page. This information is stored in a database at the back end and should be displayed afresh for every request. In this case, you can allow the entire page to be cached except for the control in which you display the count. The contents of that control will be separately generated on the server dynamically, and the rest of the page will be retrieved from the cache. This technique is called cache substitution.

In ASP.NET, you can implement cache substitution by using any of the following methods:

- Using the Substitution control
- Using the `WriteSubstitution()` method
- Using the AdRotator control

To use the Substitution control for cache substitution, add a Substitution control to the page by using the declarative or programmatic syntax. The declarative syntax is as follows:

```
<asp:substitution id="substVisitCount"
methodName="GetUserVisitCount" runat="Server">
</asp:substitution>
```

Note that the `methodName` attribute is set to the name of a static callback method defined in the page's code-behind file. The method is automatically called when the control is executed and should be defined so that it accepts an instance of `HttpContext` class and returns a string indicating the updated data to be displayed on page in place of the control. This is the signature for the `System.Web.HttpResponseSubstitutionControl` delegate that will be used to handle the callback method. The following code shows the definition for the sample callback method:

```
public static string GetUserVisitCount (HttpContext context)
{
    // Code to return count as a string
}
```

Programmatically, you can use the `System.Web.UI.WebControls.Substitution` class and its members.

To add more flexibility to this substitution technique, you can use the `WriteSubstitution()` method of the `HttpResponse` class. This will also ensure that content can be generated dynamically without the need to associate a static callback method of the page. The `WriteSubstitution()` method accepts the name of a method that will generate the content dynamically. This method whose name is passed as an argument must have the same syntax as the callback method in the case of the Substitution control. However, it need not be static and can also be associated with any object other than the page.

TAKE NOTE *

By using the Substitution technique (declarative or programmatic), you can convert client-side cacheability to server-side cacheability.

The third way to implement cache substitution is by using an AdRotator control, which is used to incorporate advertisement banners on a page and has built-in cache substitution. When you use this control, it will automatically be loaded dynamically on every request, regardless of whether the page is cached. Note that by default, pages containing this control are cached on the server.

X REF

For more information on the AdRotator control, refer to Lesson 2, “Creating and Configuring Server Controls.”

Understanding Fragment Caching

You may not always need to cache entire pages but only small parts of them. In that case, you can include the parts to be cached in user controls and set up caching for those controls. This is called ***fragment caching***.

To set up output caching for a user control, you use the `@OutputCache` directive in the `.ascx` file. The attributes of this directive that are not applicable in a `.ascx` file are `Location`, `VaryByHeader`, `CacheProfile`, and `NoStore`. An additional attribute that is applicable only for fragment caching but not page output caching is `Shared`. You can set this attribute to `true` to allow multiple web pages to share the cached version of the user control. If this is set to `false`, a separate version of the user control will be cached for each page containing it.

Cache substitution can also be used in user controls.

You can also implement fragment caching programmatically by using the `PartialCaching` attribute. Optionally, you can also use the `ControlCachePolicy` class to define the cache related settings.

Understanding Application Data Caching

In addition to caching page output, you can also cache application data by storing it in memory. This is possible in ASP.NET through the members of the `System.Web.Caching` namespace, particularly the `Cache` class. This process is known as ***application data caching***.

The `Cache` class simplifies the use and implementation of application data caching by providing access to data stored in cache in the form of key-value pairs. This class provides an application programming interface (API) to explicitly add, retrieve, or remove any data to/from a cache in memory. You can also control when items in the cache expire or are invalidated based on other dependencies. To understand these features, let us discuss the `Cache` class in detail.

USING THE CACHE CLASS

A single instance of the `Cache` class is created for every application domain and can be accessed through the `Cache` property of the `Page` object (or `HttpContext` object's `Cache` property). The commonly used properties of the `Cache` class include:

- **Item:** Sets the value of a data item in the cache. The name of the data item is passed as an argument to the property, and usage of this property is in the form of `Cache("keyname") = "value"`. Here, `Cache` is the property of the `Page` object referring to the `Cache` instance for the application and the `keyname` is the name of an item in the cache.
- **Count:** Retrieves the number of items in the cache for the application.
- **EffectivePercentagePhysicalMemoryLimit:** Defines the percentage of the computer's memory that is available to the application for caching. Another similar property is `EffectivePrivateBytesLimit`, which retrieves the number of bytes that are available for caching. You can use these properties to track the number of items that you should add to your cache before ASP.NET starts clearing items from the cache due to low availability of space. Scavenging is the process in which ASP.NET starts clearing the cache. The cache algorithm used by ASP.NET is written so that it removes the low-priority or least-used cache items first.

TAKE NOTE*

You can set the preceding two size-related properties of the cache and other such attributes in the `web.config` file under the `caching` element. These configuration settings are applicable at an application level. The `cache` child element and its attributes such as `privateBytesLimit` are used to make these settings. For information, refer to MSDN.

The commonly used methods of the `Cache` class include `Add`, `Insert`, and `Get`.

USING THE ADD () METHOD

You can use this method to add an item to the cache. The method accepts the following arguments:

- Keyname of the item as a string value.
- Value (data to be cached) of the item as an Object type.
- Dependencies on any file, cache item, or SQL Server data as a `System.Web.Caching.CacheDependency` type. Note that you cannot use the `Item` property to set a value of a key with a dependency defined for it. You will learn about cache dependencies shortly in this lesson.
- Expiration in absolute time as a `DateTime` type.
- Sliding expiration as a `TimeSpan` type. When you want to set an absolute expiration, you should set the sliding expiration to `NoSlidingExpiration` field of the `Cache` class. Similarly, if a sliding expiration is used, you should set the absolute expiration to `NoAbsoluteExpiration` field of the `Cache` class.
- Priority of the item as a `System.Web.Caching.CacheItemPriority` enumeration value. The value is instrumental during scavenging as explained earlier. The members of the enumeration are `Low`, `BelowNormal`, `Normal`, `AboveNormal`, `High`, `NotRemovable`, and `Default`.
- The name of a method that is invoked when the item is deleted from the cache. The signature of the method specified is the same as that of the `System.Web.Caching.CacheItemRemovedCallback` delegate. The method will accept the name and value of the item as the first two arguments. The third argument will be a value from the `System.Web.Caching.CacheItemRemovedReason` enumeration that will specify the reason for the removal. The members of this enumeration are `Removed`, `Expired`, `Underused`, and `DependencyChanged`.

USING THE INSERT() METHOD

You can use this overloaded method to insert a new item in the cache or update an existing item in the cache. Unlike the `Add()` method, the `Insert()` method will not fail on an attempt to insert an item if the specified item already exists. Instead, it will update the value of that item. You can use the various overloaded forms of the method, such as `Insert(String key, Object value)`, `Insert(String key, Object value, CacheDependency)`, `Insert(String, Object, CacheDependency, DateTime, TimeSpan, CacheItemPriority, CacheItemRemovedCallback)`. For details, refer to MSDN.

USING THE GET() METHOD

You can use the `Get()` method to obtain data from the cache. The data is retrieved by passing the keyname of the item as an argument of this method. The data is returned as an `Object` type. If the key is not found in the cache, a null reference is returned. Similarly, you can use the `Remove()` method to remove the item, whose key is passed as an argument, from the cache.

TAKE NOTE *

You can remove items using the `Remove()` method where the remove callback specified in the `Add()` or `Insert()` method is called to notify the user or application and perform a desired action. However, when ASP.NET itself removes an item, it uses the `CacheItemUpdateCallback` delegate. For information on this delegate, refer to MSDN.

Before you look at a cache implementation example, let us discuss another crucial class that is used to establish the dependency of cache items on different sources.

ESTABLISHING DEPENDENCIES

The `CacheDependency` class can be used to set up a dependency of a cache item on files or other cache items or another cache dependency and also invalidate that item when the source changes. The instance of the `CacheDependency` class is created to set up a dependency. You can use any of its overloaded constructors to instantiate the class.

The properties of this class include:

- **HasChanged:** Determines if the source has been updated and whether the `CacheDependency` instance has changed. A value of `true` indicates that the instance

MORE INFORMATION

For detailed information about other members of the class and the overloaded constructors of this class, refer to MSDN.

MORE INFORMATION

For detailed information on configuring SQL Server and applications for SQL Server cache dependencies, refer to MSDN.

has changed and therefore the item has been invalidated. In that case, you need to write appropriate code to generate the item afresh and add it to the cache again.

- **UtcLastModified:** You can retrieve the value of this property to identify the UTC time of the last change to the **CacheDependency** instance.

If you want cache items to be invalidated based on monitoring different dependencies, you can use the **AggregatorCacheDependency** class. This class tracks changes to a set of **CacheDependency** instances and invalidates the cache item when any of them is updated. Because the **AggregatorCacheDependency** class is derived from the **CacheDependency** class, you can pass an instance of the **AggregatorCacheDependency** class to the **Insert()** or **Add()** methods to associate the item with a collection of dependencies.

In addition to file dependencies, you can also set up data dependencies on data stored in SQL Server. This is crucial for applications that retrieve and store data in a SQL Server database. However, for this, you first need to configure SQL Server and then implement SQL Server dependencies in your program by using the **SqlCacheDependency** and **SqlCacheDependencyAdmin** classes. The **SqlCacheDependency** class allows you to add/remove/modify those items in cache that have a dependency on a table or a query of a SQL Server database. The properties and methods are similar to those of the **CacheDependency** class. The **SqlCacheDependencyAdmin** class is used to programmatically configure SQL Server databases and tables to handle notifications related to SQL Server cache dependencies. You can configure a SQL Server cache dependency through the **SqlCacheDependency** child element of the caching element.



USE APPLICATION DATA CACHING

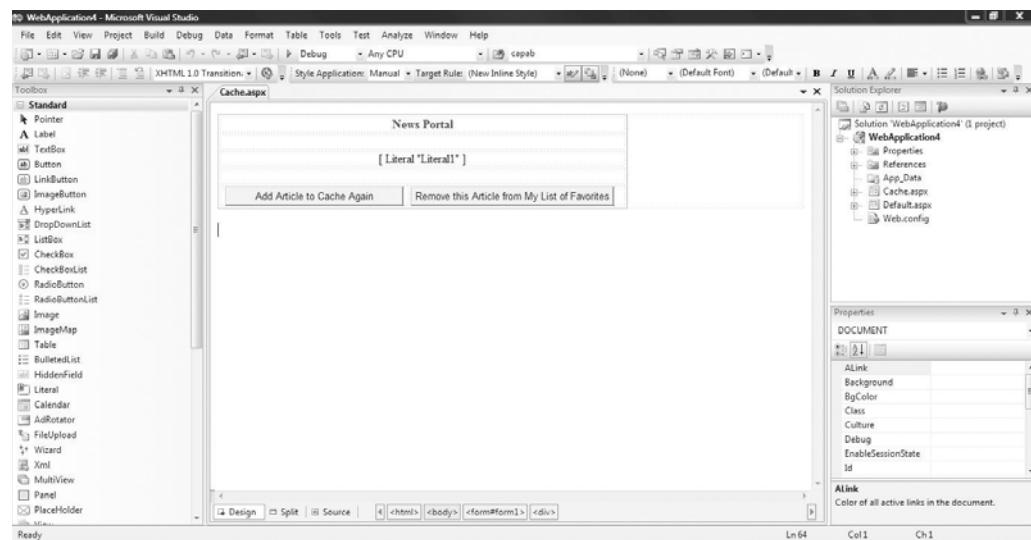
Suppose you are developing an online news portal. You want the articles to be chosen from the text file and displayed in web pages. You also want to cache the article web pages until the text in the text file changes. In addition, you want to provide the user with an option to explicitly remove an item from cache after it has been read.

To do so:

1. First design a form where the news article will be displayed as shown in Figure 9-1.

Figure 9-1

Sample web site



Note that the Add Article to Cache Again button has been included only for demonstrative purposes to show you how the article is added to cache when the user clicks this button.

2. Then in the code behind file, add the System.Web.Caching namespace in addition to any other namespaces required. Now, add the following code to the code-behind file:

```
public partial class Cache: System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        AddItemToCache();
    }
}
```

3. To implement the desired functionality, at page load, a method AddItemToCache() should be invoked to add the news article read from the file to the cache if it is not already in the cache. The following code shows the definition of this method:

```
public void AddItemToCache()
{
    CacheDependency dep1 = new
    CacheDependency(Server.MapPath("cache.txt"));
    if (Cache["News"] == null)
    {
        string key = Getvalue();
        Cache.Insert("News", key, dep1,
        DateTime.Now.AddSeconds(120),
        System.Web.Caching.Cache.NoSlidingExpiration,
        CacheItemPriority.High, NewsRemove);
        Literal1.Text = Cache["News"].ToString();
    }
    else
    {
        Literal1.Text = Cache["News"].ToString();
    }
}
```

In the preceding code, the method forms a dependency of the cache on a file so that any updates to the file invalidate the cache. Then, if the cache has an item named News, the article data is read from the cache and set as the value of the Text property of the Literal1 control. However, if the cache item named "News" is not found in the cache then it is read from the file by using the GetValue() method:

```
private string Getvalue()
{
    string fName = Server.MapPath("cache.txt");
    StreamReader testTxt = new StreamReader(fName);
    string allRead = testTxt.ReadToEnd();
    testTxt.Close();
    return allRead;
}
```

4. Then, the item is inserted into the cache using the Insert() method of the Cache class.
5. The expiration is used as absolute and set to 2 minutes, and the priority is set to high. The file dependency dep1 is also passed an argument to the Insert() method. Note that NewsRemove is the name of the method to be called when the item is removed from the cache.

6. Also add the following declarations to your class:

```
static bool NewsRemoved = false;
static CacheItemRemovedReason Newsreason;
CacheItemRemovedCallback NewsRemove = null

protected void Button1_Click(object sender, EventArgs e)
{
    Cache.Remove("News");
    AddItemToCache();
}
```

The first variable will be set when the data is removed from the cache. The second variable will hold a reason for removal of data from the cache. The third variable is a CacheItemRemovedCallback delegate type that will be used to invoke the callback method. The Click event handler removes and adds the article afresh to the cache when the user clicks the Add Article to Cache Again button. Now, article data caching has been implemented.

7. Next, handle cache invalidation based on the dependency so that when the data in the file changes, it will be read from the file instead of the cache and the cache will be updated. To handle the invalidation, capture the reason and display it to the user by adding the following code at the beginning of the AddItemToCache() method:

```
if (Newsreason.ToString() == "DependencyChanged")
{
    Response.Write("<BR>");
    Response.Write("Reason: <B>" +
Newsreason.ToString() + "</B>" + " " +"The Article has been
updated");
    Response.Write("<BR>");
    Response.Write("<BR>");
}
```

8. Finally, handle the explicit removal of the item when the user clicks on the Remove this Article from My List of Favorites button. This can be done associating the following handler with the Click event of the button:

```
public void RemoveItemFromCache(Object sender, EventArgs e)
{
    if (Cache["News"] != null)
    {
        Cache.Remove("News");
        Literal1.Text = " Article is not being cached
anymore because you have removed it from your favorites list";
    }
    else
    {
        Literal1.Text = "No article is found in Cache";
    }
}
```

9. In addition, add the following code at the beginning of the AddItemToCache() method to handle the removal:

```
NewsRemoved = false;
NewsRemove = new CacheItemRemovedCallback(this.RemovedCallback);
```

The definition of the callback method is as follows:

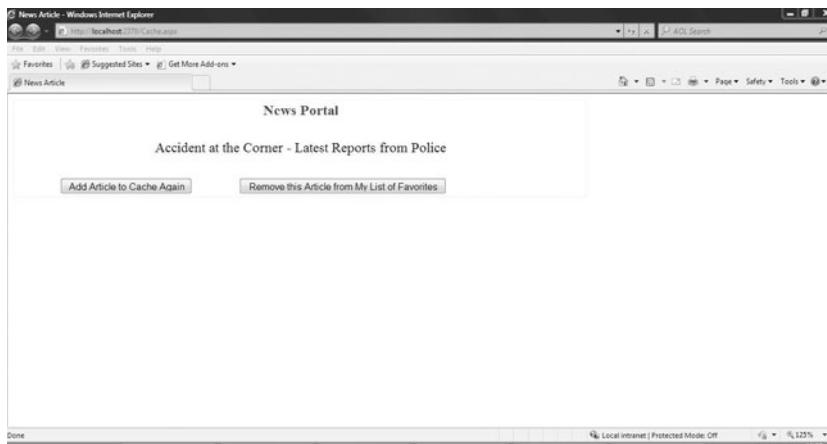
```
public void RemovedCallback(String news, Object
vnews, CacheItemRemovedReason newsitems)
{
```

```
NewsRemoved = true;
Newsreason = newsitems;
}
```

Figure 9-2 shows the page output when it is first opened.

Figure 9-2

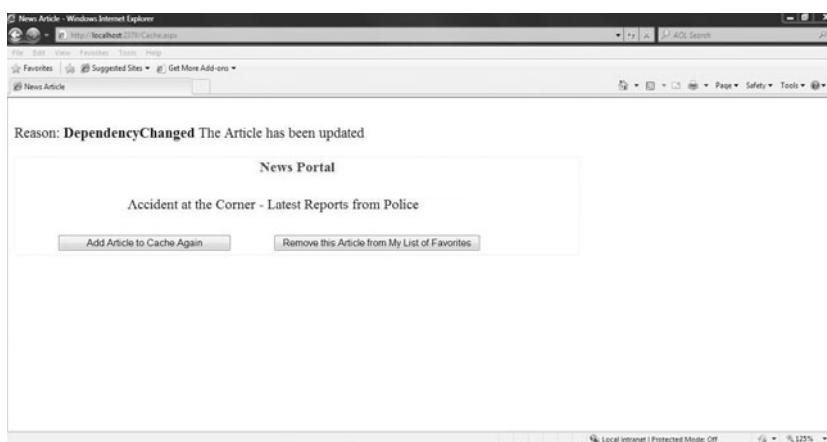
Page output



Now, if the content is updated in the text file and the page is refreshed, Figure 9-3 shows the output.

Figure 9-3

Output page on refresh



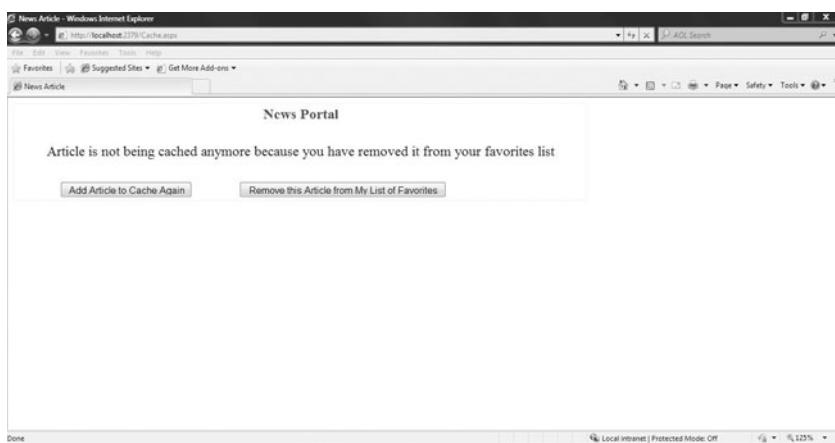
CERTIFICATION READY?

Implement session state, view state, control state, cookies, cache, or application state.
7.5

Figure 9-4 displays the output when you click the Remove this Article from My List of Favorites button.

Figure 9-4

Output on removing page from favorites



■ Securing Web Applications



THE BOTTOM LINE

Web applications are vulnerable to security attacks from various points. Therefore, when developing a web application, you need to be careful about the security of the web application to prevent compromising the security of the data and resources used by the web application.

You have already learned about a crucial security technique in the form of input validation. However, three other crucial techniques that are imperative for the security of web applications are **authentication**, **authorization**, and process **impersonation**.

In this section, you will learn how ASP.NET facilitates and simplifies the implementation of authentication, authorization, and process impersonation.

Implementing Authentication

Authentication refers to validating the identity of a user accessing a web application. You would have observed this mechanism when you are asked to log on to a Web site to access your email or other data.

ASP.NET supports authentication through the following features:

- Support and integration for different authentication modes
- Support for user management through memberships
- Server controls for login functionality

USING AUTHENTICATION MODES

ASP.NET has a built-in basic authentication code in the form of authentication providers. Based on the different authentication providers, ASP.NET supports the following types of authentication modes:

- **Windows authentication:** Uses the Windows user account (domain, username, and password) to authenticate a user in integration with IIS. It uses any or a combination of the authentication schemes supported by IIS, such as Anonymous, Basic, Digest, or

Integrated Windows. For security reasons, ideally Anonymous should be disabled if you want to use Windows authentication mode for your ASP.NET applications. This form of authentication is suitable for intranet applications and on Internet Explorer only. This mode is handled by the `System.Web.Security.WindowsAuthenticationModule` provider class. The `Authenticate` event of this class can be handled to implement any custom code for authentication.

- **Forms authentication:** Uses credentials supplied by a user on a Login page to authenticate the user against valid credentials stored in a data store. Once the user enters valid credentials and is logged in, a state management technique such as cookie or query string stores the credentials to indicate that the user has been authenticated. The cookie can be a persistent cookie or be available only till the current session lasts. This mode is handled by the `System.Web.Security.FormsAuthenticationModule` provider class.
- **Password authentication:** Uses the Microsoft Passport Service or Windows Live ID to authenticate users based on credentials that can be used across all Web sites that are members of the service. This mode is handled by the `System.Web.Security.PassportAuthenticationModule` provider class.
- **None:** Indicates that custom authentication will be used.

The desired mode for an application can be set in the `Web.config` at the application root level, as follows:

```
<configuration>
  <system.web>
    ...
      <authentication mode="ModeName">
      ...
    </authentication>
    ...
  </system.web>
</configuration>
```

In the preceding code, `ModeName` can be Forms, Windows, Passport, or None. The Forms mode is the most suitable for most Internet applications.

USING FORMS AUTHENTICATION

You can use the `forms` child authentication element in the configuration file to further configure forms authentication. One `forms` element should be configured for each application running on a server. These are the commonly used (optional) attributes of the element:

- **loginUrl:** Defines the URL of the Login page to which all unauthenticated requests are redirected.
- **defaultUrl:** Defines the URL of the page to which the user is redirected after being authenticated.
- **name:** Defines the name of the authentication cookie to be used, which is set as `.ASPXAUTH` by default.
- **protection:** Defines the type of protection used for the authentication cookie. The values can be All (default), None, Encryption, or Validation. You can also set the `requireSSL` attribute of the element to `true` to transmit the cookie using SSL for more security.
- **timeout:** Defines the expiry time in minutes for the cookie. The time can be absolute or sliding based on whether the `slidingExpiration` attribute is set to `true` or `false`.
- **cookieless:** Defines whether cookies or query strings are used. The values can be `UseCookies`, `UseUri`, `AutoDetect`, `UseDeviceProfile` (default).

One `forms` child element is `credentials`. This element allows you to configure user credentials in the configuration file and has one mandatory attribute `passwordFormat`

that defines the type of encryption used to store the password. The attribute can be set to `Clear` (no encryption), `MD5`, or `SHA1`(default). An important child `credentials` element is `user`, which represents a user credential definition added to the credentials collection. The `user` element has two mandatory attributes `name` and `password`.

The following example shows how the `forms` element can be configured:

```
...
<authentication mode="Forms">
  <forms name="mycookie" loginUrl="/login.aspx"
    cookieless="UseCookies" defaultUrl="Home.aspx">
    <credentials passwordFormat = "MD5">
      <user name="Peter" password="pwd1"/>
    </credentials>
  </forms>
</authentication>
...
```

Note that `credentials` can also be used as the child element of authentication for the Windows mode.

You can retrieve the configuration settings for the `forms` element in your program by using the helper `System.Web.Security.FormsAuthentication` class. Its static properties will retrieve the values of the settings, such as `LoginUrl`, `FormsCookieName`, `CookieMode`, and so on. For the complete list of properties, refer to MSDN.

The static methods of this class are also useful when implementing forms authentication. Some of the commonly used methods are:

- **`Authenticate()`**: Validates the credentials entered by a user against those stored in the configuration file. This method accepts the username and password input by the user as string argument and returns `true` or `false` based on the validation result.
- **`RedirectFromLoginPage()`**: Redirects a user to a requested URL after the user has successfully logged on to the site. One overloaded form accepts the URL of the page that the user originally requested, or the default URL if the former is not available, and a Boolean value indicating whether a persistent cookie has to be created. The other overloaded form accepts a third argument also defining the path of the cookie.
- **`RedirectToLoginPage()`**: Redirects a request to the Login page if the user is not authenticated. One form of the method accepts no arguments, and the other overloaded form accepts a query string to be appended to the URL to which you are redirecting. You can use the query string to show a custom error message.
- **`SignOut()`**: Removes the authentication credentials from the cookie to sign the user out of the site.
- **`SetAuthCookie()`**: Adds an authentication ticket containing the username to be stored in a cookie or query string. The method has two overloaded forms. One form accepts the username and a Boolean value indicating whether a persistent cookie has to be created. The second form accepts a third argument also defining the cookie path or the URL for the query string.

The following example demonstrates the use of the `Authenticate()` method for this configuration:

```
<authentication mode="Forms">
  <forms loginUrl="/Default.aspx" cookieless="UseCookies"
    defaultUrl="Default.aspx">
    <credentials>
      <user name="Peter" password="pwd1"/>
    </credentials>
  </forms>
</authentication>
```

MORE INFORMATION

For a complete list of methods of the `FormsAuthentication` class, refer to MSDN.

If you enter the username and password from the user on a form in text boxes named txtName and txtPass, then you can call the `Authenticate()` method in the `Click` event handler of a Login button on the form as follows:

```
if (FormsAuthentication.Authenticate(txtName, txtPass))
{
    Response.Redirect("createuserwizard.aspx");
}

else
{
    lblMessage.Visible = true;
    lblMessage.Text = "Invalid User";
}
```

USING USER IDENTITIES

When you use authentication, you may need to retrieve the identity of the current user under which the code is executing. These identities are represented by the classes `System.Security.Principal.GenericIdentity` and `System.Security.Principal.WindowsIdentity`.

Some important properties of the `GenericIdentity` class are `Name` representing the username, `IsAuthenticated` indicating whether the user has been authenticated, and `AuthenticationType` indicating the authentication mode used.

A Windows user is represented by the `WindowsIdentity` class. Some of the important properties of this class are:

- **IsAnonymous:** Checks whether the user is anonymous.
- **IsAuthenticated:** Checks whether the user is authenticated.
- **IsGuest:** Checks whether the user is a logged in through a guest account.
- **IsSystem:** Checks whether the user is a logged in through a system account.
- **Name:** Retrieves the username (Windows logon name) for the current user.
- **Groups:** Retrieves the groups to which the current user belongs. The value of the property is a `System.Security.Principal.IdentityReferenceCollection` type.
- **AuthenticationType:** Defines the type of authentication used for the user.

An important method of this class is `GetCurrent()` that returns the current user. It has overloaded forms and also has some related methods for impersonation that are discussed later in this lesson.

IMPLEMENTING ASP.NET MEMBERSHIP SUPPORT FOR AUTHENTICATION

The membership feature in ASP.NET encapsulates all functionality related to form authentication from creation of user accounts to authentication of users, management of passwords, and storage of and access to user account information on the back end. It greatly simplifies implementing form authentication mode in your ASP.NET application by using **membership providers**, which are the modules that actually contain all the code needed to implement the membership feature and interface between the stored membership data and the forms for authentication. They manage where and how membership data is stored. ASP.NET supports two built-in membership providers, `ActiveDirectoryMembershipProvider`, which uses Microsoft Active Directory Service to store membership data and `SqlMembershipProvider` (default), which uses SQL Server to store membership data. In addition, you can define custom membership providers by extending the `System.Web.Security.MembershipProvider` class and using them for authentication.

MORE INFORMATION

For more information about creating custom providers, refer to MSDN.

TAKE NOTE *

The authentication mode must be set to form authentication before you configure and use membership.

In order to implement authentication through membership in your web application, you first need to configure membership-related settings and then create forms for user creation, login, password management, and so on. The latter step has been largely simplified through the introduction of the login controls, which will be discussed in the next topic. These controls automatically use the membership feature. You can also use the Web Site Administration Tool to create users.

In order to configure an application for membership, you can make entries in the configuration file. Choose your membership provider, and add it to the web.config file as follows under the `<system.web>` element:

```
<membership defaultProvider="providername" attributes....>
    <providers>
        <add name="providername" type="providerType"
            connectionStringName="connectionString to provider"
            applicationName="appName" (...) />
        ...
    </providers>
</membership>
```

In the preceding code, the `<membership>` element is used to set configuration for the membership feature. Three optional attributes of this element are:

- **defaultProvider**: Specifies the provider to be used by default. When you install ASP.NET, the `SqlMembershipProvider` is set as the default in `machine.config`.
- **userIsOnlineTimeWindow**: Specifies the time (in minutes) since the last user activity related to a user account for which the user is considered to be online. If there is no activity after this period elapses, the user is logged out.
- **hashAlgorithmType**: Specifies the encryption algorithm to be used for encrypting password values by the membership providers that are configured to use a hashed password.

The child element `<providers>` represents the collection of membership providers, while its child element `<add>` is used to add membership providers. The `<add>` element has various attributes, such as:

- **name**: Set the name of the provider here.
- **type**: Set the fully qualified classname representing the type of provider here.
- **connectionStringName**: Set the name of the connection string corresponding to the data store used by the specified provider here. The connection string is defined in the `<connectionString>` element of `web.config`.
- **applicationName**: Set the name of your application here.
- **enablePasswordRetrieval**: Set this to `true` to allow the original password to be recovered.
- **enablePasswordReset**: Set this to `true` to allow the original password to be reset.
- **requiresQuestionAndAnswer**: Set this to `true` to enable security question-answer functionality to be included mandatorily during account creation or password recovery.
- **requiresUniqueEmail**: Set this to `true` to require a unique email address from the user, for example during account creation.
- **passwordFormat**: Set this when using the SQL Server provider to define how the password is stored. The value can be a member of the `MembershipPasswordFormat` enumeration, which can be `Clear`, `Hashed`, and `Encrypted`.
- **maxInvalidPasswordAttempts**: Set this to an integer value to define the number of invalid password-answer entry attempts allowed. The default value is 5.
- **minRequiredPasswordLength**: Set this to an integer to define the minimum number of characters allowed in the password. The default value is 1.
- **passwordStrengthRegularExpression**: Set this when using the SQL Server provider to define a regular expression for the password.

The following code shows an example of the configuration for the default SQL Server provider (using the connection string `SqlCon`) for an application named `newsportalApp`:

```
...
<connectionStrings>
    <add name="SqlCon" connectionString="server=.;database=aspnetdb;
        integrated security=sspi;" />
```

MORE INFORMATION

For a complete list of the attributes on the `<add>` element, refer to MSDN.

TAKE NOTE*

To use the SQL Server provider, you must ensure that the membership database (default is aspnetdb) is installed.

```
</connectionStrings>
...
<membership defaultProvider="MyMemSqlProvider"
userIsOnlineTimeWindow="5">
  <providers>
    <add name="MyMemSqlProvider"
    type="System.Web.Security.SqlMembershipProvider"
    connectionStringName="SqlCon" applicationName="newsportalApp"
    enablePasswordRetrieval="true" enablePasswordReset="false"
    requiresQuestionAndAnswer="false" requiresUniqueEmail="true"
    passwordFormat="Clear" />
  </providers>
</membership>
```

ASP.NET also provides an API to use the membership and membership provider through various classes such as `Membership` and `MembershipUser`.

The `Membership` class provides the API for authentication of the user through login credentials. In addition, it also allows you to store and manage the membership data.

The class provides various properties and methods for these functions. The properties are used to retrieve the settings made in the configuration file for further processing based on their values. Some of the properties include `ApplicationName`, `Provider` (corresponds to default provider), `MaxInvalidPasswordAttempts`, `EnablePasswordReset`, and so on.

The commonly used static methods of the class are:

- **CreateUser()**: Creates a new user account and includes various overloaded forms, two of which include: `CreateUser(string name, string pwd)` that accepts only the username and password, and `CreateUser(string name, string pwd, String email, string question, string answer, bool approved, Object providerUserKey, out MembershipCreateStatus status)` that accepts the username, password, security question, answer to the security question, true or false to indicate whether the user can log on to the site after account creation, an identifier for the user, and an account status that is a member of the `MembershipCreateStatus` enumeration. The method returns an instance of the `MembershipUser` class.
- **DeleteUser()**: Deletes a user account from the data store. This method has two overloaded forms: `DeleteUser(string username)` that accepts the username of the account to be deleted, and `DeleteUser(string username, bool del)` that accepts a Boolean value indicating whether the data of the user account should be deleted (`true`) or retained (`false`).
- **GeneratePassword()**: Generates a new random password during password recovery. The length of the password and minimum number of non-alphanumeric characters that should be in the password are passed as arguments. The password is returned as a string value.
- **GetUser()**: Retrieves data related to the user account. It has various overloaded forms. Two commonly used forms are `GetUser(string username)` that retrieves information for the username passed as an argument, and `GetUser(Object id, bool update)` that retrieves information for the user account corresponding to the ID passed as the first argument. The second argument should be set to `true` to update the date/time for the last activity for the retrieved user account. This method returns an instance of the `MembershipUser` class.
- **ValidateUser()**: Authenticates the user whose username and password are passed as arguments to this method. The method returns `true` or `false` depending on the status of validation.
- **UpdateUser()**: Updates user account data by passing an instance of the `MembershipUser` class to the method. The instance stores the updated data.

CERTIFICATION READY?

Configure providers.

1.1

MORE INFORMATION

For more information on the `MembershipUser` class and other classes that provide an API for membership features, refer to MSDN.

Understanding Server Controls for Login Functionality

ASP.NET has simplified the authentication through login credentials greatly by providing Web server controls that may be used without the need for any programming. The login controls work using the HTTP protocol and use plain text for the login credentials. To increase security, you can configure them to use the secure version of HTTP, which is HTTPS with Secure Socket Layer (SSL). Let's discuss the main controls.

The main controls include:

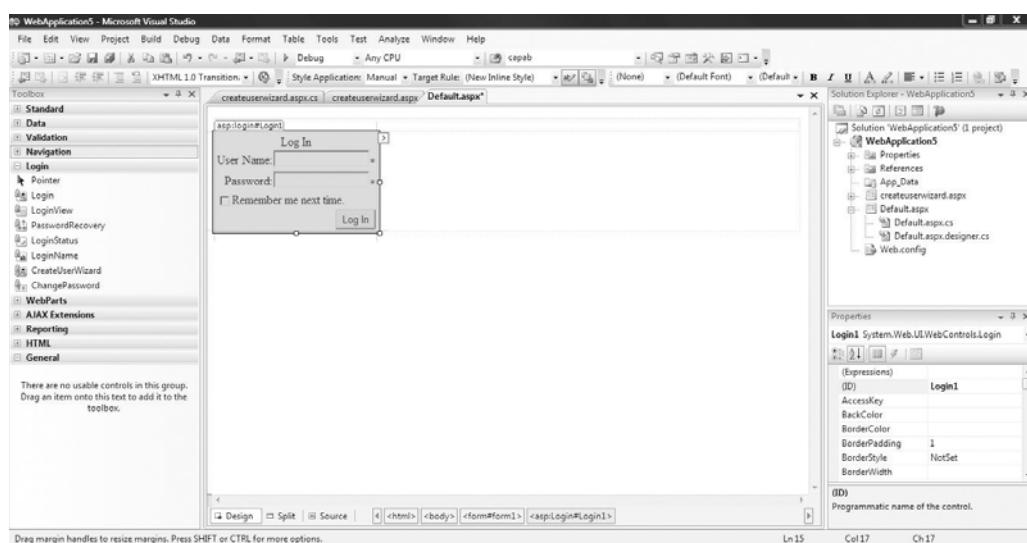
- Login control
- LoginName control
- LoginView control
- LoginStatus control
- PasswordRecovery control
- CreateUserWizard control
- ChangePassword control

USING THE LOGIN CONTROL

The Login control represents a complete UI that allows the user to log on to a web application. Figure 9-5 shows the components of a Login control, a label, two text boxes for user-name and password, a check box for enabling automatic log on functionality on subsequent visits, and a button to submit the login credentials.

Figure 9-5

Components of Login controls



The control has been dragged and dropped from the Login section in the Toolbox, which is also visible in Figure 9-5. This control can be customized as desired by setting its properties. You can add various other elements, such as a Register New User link, a Help link, or even customized error messages on login failure.

Some of the important properties of the `System.Web.UI.WebControls.Login` class are:

- **Username field-related properties:** You can use the `UserName` property to retrieve the username entered in the text box within the Login control. You can also customize the text displayed in the label next to the text box for User Name by setting the `UserNameLabelText` property. In addition, you can set the

`UserNameRequiredErrorMessage` to define the custom error message text displayed if the user does not enter any value for the username. The error message is displayed in the `ValidationSummary` control. Note that input validation is already built into the `Login` control through the validation controls associated with the controls within the `Login` control.

- **Password field-related properties:** You can use the `Password` property to retrieve the password entered in the text box within the `Login` control. The `PasswordLabelText` and `PasswordRequiredErrorMessage` properties perform similar functions to the `LabelText` and `RequiredErrorMessage` properties of the username field. You can also specify a hyperlink to a page from which the user can recover the password. This is done by setting the URL of the page and text of the hyperlink using the `PasswordRecoveryUrl` and `PasswordRecoveryText` properties. You can also set an icon to be displayed against the hyperlink by setting the `PasswordRecoveryIconUrl`. You can also set the URL, text, and icon to incorporate the help feature by setting the `HelpPageUrl`, `HelpPageText`, and `HelpPageIconUrl` properties. Similarly, a new user registration functionality can be included by setting the `CreateUserUrl`, `CreateUserText`, and `CreateUserIconUrl` properties.
- **Remember Me checkbox related-properties:** You can customize the default text displayed against the check box on the `Login` control by setting the `RememberMeText` property. You can also enable automatic login by storing a cookie with an expiration of 50 years on the client and by setting the `RememberMeSet` property to `true`. The cookie contains the username; so, the user does not have to explicitly log in to the site because the password is retrieved for the username in the cookie and authenticated automatically. The cookie is deleted from the client if the user specifically logs out of the Web site. You can control the `RememberMeSet` property in code and also allow the user to control it by ensuring that the Remember Me check box is displayed. You can enable or disable the display of the Remember Me check box by setting the value of the `DisplayRememberMe` property to `true` or `false`.
- **Login failure related-properties:** You can define the action to be taken if the login attempt fails by setting the `FailureAction` property. The value of this property is a member of the `System.Web.UI.WebControls.LoginFailureAction` enumeration, which includes `Refresh` and `RedirectToLoginPage`. In addition, you can also customize the text displayed when the login attempt fails by setting the `FailureText` property.
- **DestinationPageUrl:** You can set the value of this property to the page that the user is redirected to after successfully logging in. If you do not set this property, then you will either be redirected to the same page or to the default page specified in the `form` element's `defaultUrl` child element in the configuration file. By default, this URL is set as `default.aspx` in the configuration file.
- **InstructionText:** You can set a textual description to be displayed underneath the title in the `Login` control to give the user information about logging in by setting the value of this property.
- **MembershipProvider:** You can define the membership data provider used to validate the login credentials of the user. The `Login` control authenticates the user by using the default membership provider without the need for you to write any code. However, you can customize this feature. You will learn more about membership later in the lesson.
- **Orientation:** You can define whether the child controls of the `Login` control are oriented vertically (default) or horizontally by setting this property.
- **VisibleWhenLoggedIn:** You can control whether the `Login` control is displayed or hidden after a successful login by setting this property.

MORE INFORMATION

The `System.Web.UI.WebControls.Login` class has other properties such as `LoginButtonText` and `TitleText`. For a complete list, refer to MSDN.

In addition to these properties, there are other style-related properties. The following code example demonstrates the usage of some of the preceding properties and some style-related properties:

```
<asp:Login ID="lgnNewsPortal" runat="server" CreateUserText="New User??" CreateUserUrl="~/register.aspx" FailureText="Login has failed!"
```

```

Try Again." InstructionText="Login if you are an existing user or
create a new account to view news articles." PasswordRecoveryText="For
got Password???" PasswordRecoveryUrl="~/Recover.aspx">
<CheckBoxStyle BorderStyle="Double" ForeColor="Red" />
<InstructionTextStyle Font-Bold="True" Font-Italic="True"
ForeColor="Red" />
<FailureTextStyle BackColor="#3366FF" />
</asp:Login>

```

The output of the Login control defined using this code is displayed in Figure 9-6.

Figure 9-6

Output Login control

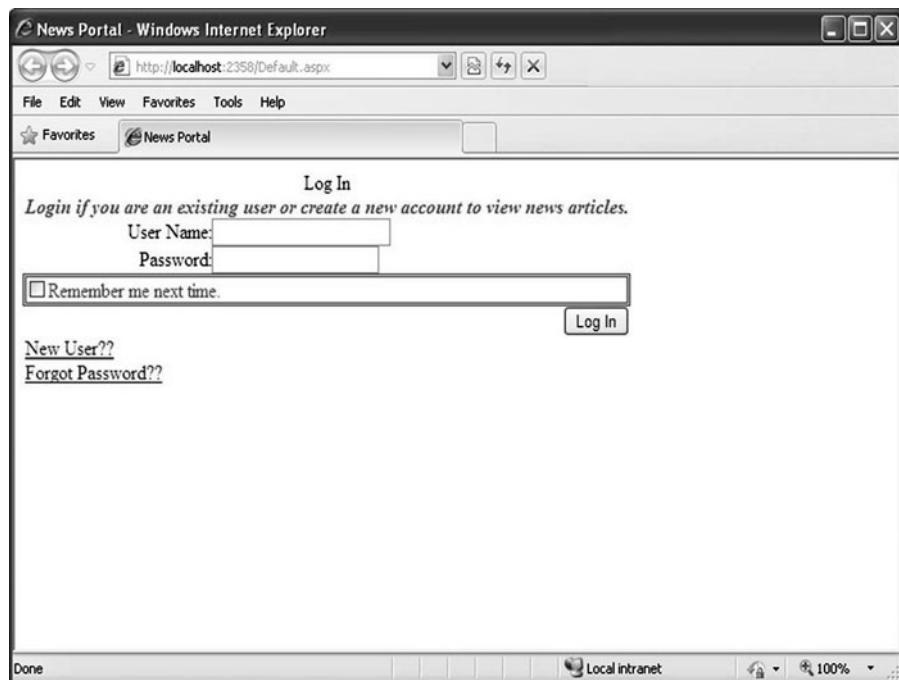
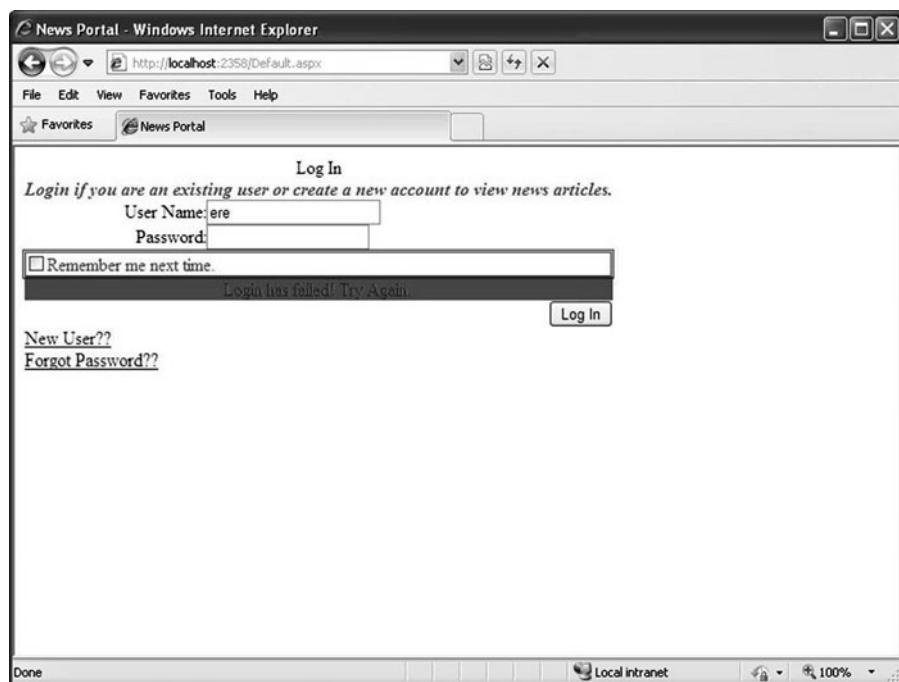


Figure 9-7 shows the page on login failure. Note the appearance and content of the custom message displayed.

Figure 9-7

Result of Login failure



The Login class also has some important events and event handlers. The Authenticate and LoggedIn events are very important in this context.

USING THE AUTHENTICATE EVENT

This event is raised when the user is authenticated. You can define a handler for this event to implement any custom actions or logic to be executed for authentication. The `OnAuthenticate(object sender, AuthenticateEventArgs args)` method raises this event and sets the `Authenticated` property of the `AuthenticateEventArgs` class to `true` if the login is successful. If you want a custom action to be performed before the property is set to `true`, you can define a callback method as follows:

In the code-behind file:

```
protected void lgnNewsPortal _Authenticate(object sender,  
AuthenticateEventArgs e)  
{  
    // Custom authentication logic implementation code.  
    // Set the e.Authenticated to true or false based on custom logic  
}
```

In the declaration of the control:

```
<asp:Login ID="lgnNewsPortal" runat="server" ...  
OnAuthenticate= lgnNewsPortal _Authenticate>  
</asp:Login>
```

USING THE LOGGEDIN EVENT

This event is raised after the `Authenticate` event on successful login of the user to the web application. You can define a handler for this event to perform custom tasks, such as retrieve the logged in user's preferences from a database to display the home page according to those preferences.

USING THE LOGINNAME CONTROL

After a user has logged in, Web sites usually display the name of the logged in user on the pages. This can be done by adding the `LoginName` control at a desired location within the `<form></form>` tags. The name displayed is based on the authentication scheme used (ASP.NET form or Windows). The name is retrieved from the `User` property of the `Page` class.

The `LoginName` control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `LoginName` class. You can customize this control by using the properties of the class. A commonly used property of this class is `FormatString`, which is used to define the text that will be displayed along with the name of the user. The default value of the property is "`{0}`" where `{0}` indicates a placeholder for the name of the logged in user. You can modify it by adding more text and placing the placeholder at the position where you want the name to appear. For example, if you want to display "Good morning <username> !" then you should set the value of the property to "Good morning `{0}` !". If you omit `{0}`, the text will be displayed, but the name of the logged in user will not.

USING THE LOGINSTATUS CONTROL

For login purposes, many Web sites display a `Login` link to redirect the user to a `Login` page. Once a user is logged in, all pages of the Web site can display a `Log Out` link to allow the user to log out of the Web site. In ASP.NET, you incorporate this functionality easily by using the `LoginStatus` control.

The `LoginStatus` control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `LoginStatus` class. Some commonly used properties of this class are:

- **LoginText:** Customizes the text displayed in the login hyperlink, which is "Login" by default. This property is similar to the `LogoutText` property.

- **LoginImageUrl:** Displays an image for the login hyperlink instead of the text when you set this property to the URL of the image. The login text is then used as the alternate text for the image. This property is similar to the **LogoutImageUrl** property.
- **LogoutAction:** Defines the action to be taken when a user logs out of the Web site by clicking on the link displayed by this control. The value of the property can be any member of the `System.Web.UI.WebControls.LogoutAction` enumeration, which includes `Refresh` (default), `Redirect`, and `RedirectToLoginPage`. If the value is set as `Redirect`, you can use the `LogoutPageUrl` to define the URL that the user should be redirected to when they click the Log Out link.

ASP.NET completely handles the feature implemented by this control, even to the level of clearing all view state and postback data, so that it is not available to anyone after logging out. Moreover, if a cookie is being used to store the credentials of the user for automatic login, then that is also deleted, and user will be asked to log in the next time they open the Web site.

Crucial events of the class that you can handle are `LoggedOut` and `LoggingOut`. The `LoggedOut` event occurs after the user has clicked the Log Out link displayed by the control and has been logged out. When this event is raised all state data is lost. The method is raised by the `OnLoggedOut(EventArgs arg)` method.

The `LoggingOut` event is raised just after the user clicks the Log Out link but before the user is actually logged out. The method is raised by the `OnLoggingOut(LoginCancelEventArgs arg)` method. Handling this event can be useful when you want to perform some processing before you allow the user to log out. For example, if a user is editing his account settings just before clicking the Log Out link, you can handle the event and confirm whether the user wants to log out without saving the changes made to the account.

USING THE LOGINVIEW CONTROL

The appearance of content on many Web sites varies before logging in and after logging in. Moreover, some sites provide different types of memberships and vary the look and feel of the UI and content based on the type of user that has logged in. This type of functionality can be achieved by using the `LoginView` control.

The `LoginView` control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `LoginView` class. You can customize the appearance for each type of user by setting the properties of the `LoginView` class:

- **AnonymousTemplate:** Set this property to a template (an implementation of `ITemplate` interface) to display content in a specific way to users who are not logged in to the Web site.
- **LoggedInTemplate:** Set this property to a template to display content in a specific way to users who have successfully logged in to the Web site.
- **RoleGroups:** Set this property when you want to use different templates to display content for logged in users having different roles. The value of the property is a `System.Web.UI.WebControls.RoleGroupCollection` type. The `LoggedInTemplate` property value is used for users who are logged in but are not members of any of the specified role groups. You will learn about roles later in the lesson.

Some useful events that you can handle for customization include:

- **ViewChanged:** This event is raised after the `LoginView` control has changed from a template for one type of user to a template for another type of user as a result of the change in the login status of a user during postback. The event is raised by the `OnViewChanged(EventArgs args)` method. You can handle this event to perform custom actions, such as initialization actions for the child controls in the new view.
- **ViewChanging:** This event is raised due to a change in login status but before the view is changed by the `LoginView` control. The event is raised by the `OnViewChanging(EventArgs args)` method. You can handle this event to perform custom actions, particularly to save data in the controls in the original view.

USING THE PASSWORDRECOVERY CONTROL

The PasswordRecovery control is used to implement the password recovery through email functionality where an email is sent if the user has forgotten the password and requests that the original or a new password be sent to an email address.

The PasswordRecovery control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `PasswordRecovery` class. This control supports three views: `UserName`, `Question`, and `SuccessName`. In the first view, the username is entered and the password is recovered/reset based on the username. In the second view, if it is supported by the membership provider, the username view is first displayed, and then, if the username is valid, a security question for the user (corresponding to the username entered) is displayed and the password is recovered/reset on the basis of the answer to the question. The third view is the success view in which the user is notified whether the password reset/recover is successful.

Some of the commonly used properties of the `PasswordRecovery` class are:

- **Properties related to the control's view:** The `PasswordRecovery` control can recover/reset the password based on the username or a security question/answer. You can set a template for these by setting the `UserNameTemplate` property or the `QuestionTemplate` property. Note that the latter view must be supported by the membership provider for the login functionality in order to be displayed. You can set the `SuccessTemplate` property to define a template for the Success view. Note that different properties of the `PasswordRecovery` class are applicable in or affect the three views.
- **Properties related to Security Question view password recovery:** You can set questions/answers for password recovery, as you have seen on many emailing Web sites. This question/answer will be used if the Question view is enabled. You can retrieve the security question previously set by the user (at the time of registration) from the `Question` property. You can retrieve the answer to the security question by using the `Answer` property. You can customize the message displayed when the answer entered by the user does not match the value of the `Answer` property by setting the `QuestionFailureText` property.
- **Properties related to Username view for password recovery:** You can retrieve the username entered by the user to be used for password recovery by using the `UserName` property (in the Username view). You can also customize the message displayed when the value of the `UserName` property does not match any username on the Web site by setting the `UserNameFailureText` property.
- **Properties related to Success view for password recovery:** You can define the text message to be displayed on successfully sending the recovered/reset password by setting the `SuccessText` property. You can also specify the URL of the page that you want to redirect the user to on successful recovery/reset by setting the value of the `SuccessPageUrl` property. If you do not assign a value to this property, the page for password recover is refreshed.
- **MailDefinition:** The recovered/reset password is sent to the user in an email message. The email message related settings, such as subject, address, and so on are configured in a `System.Web.UI.WebControls.MailDefinition` instance. You can retrieve these settings by accessing this property. Note that SMTP settings must be configured for the application to allow the application to send an email.
- **MembershipProvider:** Many functions of the `PasswordRecovery` control depend on the membership provider. For example, the password can be recovered only if it is not encrypted using nonreversible encryption, in which case it is reset. This depends on the membership provider's support for clear text or encrypted passwords. Similarly, the view also depends on the membership provider. So, the membership provider to be used by the control is defined in this property.

Some useful events of the class are:

- **SendingMail:** This event is raised after the username/answer details have been entered and successfully validated but just before the user is sent the recovered/reset password by email. You can handle this event, raised by the `OnSendingMail(MailMessageEventArgs args)` method to perform processing related to storing the password or setting a custom email message and so on.
- **SendMailError:** This event is raised when an attempt to send the email fails and an error is raised by the SMTP system. The event is raised by the `OnSendMailError(SendMailErrorEventArgs args)` method and should be handled to prevent the failure from being made apparent to the user. For this, the `Handled` property of the `SendMailErrorEventArgs` class should be set to `true` in the event handler followed by the code to handle the error.
- **UserLookupError:** This event is raised when the username entered by the user for password recovery is not a valid username on the Web site. The event is raised by the `OnUserLookupError(EventArgs args)` method.
- **AnswerLookupError:** This event is raised in the Question view when the user enters a wrong answer to the security question. The event is raised by the `OnUserLookupError(EventArgs args)` method.
- **VerifyingUser:** This event is raised before sending the username to the membership provider for authentication. You can process the username before sending it for authentication by handling this event. For example, you may want to add a prefix to the username for it to match any valid username. The `OnVerifyingUser(LoginCancelEventArgs args)` raises this event. Similarly, the `VerifyingAnswer` event is raised before sending the answer of the security question to the membership provider for authentication. The event is raised by the `OnVerifyingAnswer(LoginCancelEventArgs args)` method.

MORE INFORMATION

The preceding is only a list of some commonly used properties. For detailed information about style-related settings and properties for the PasswordRecovery control, refer to MSDN.

USING THE CHANGEPASSWORD CONTROL

The ChangePassword control is used to change the password in a Web site. It works in two views: Change Password view where the user can enter the original password and then enter the new password twice, and Success view where the user is shown a confirmation that the password has been changed.

The ChangePassword control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `ChangePassword` class. Some of the commonly used properties of this class are:

- **UserName:** You can use this property to retrieve the username for which the password has to be changed. The password can be changed by the currently logged in user, or by a user when not logged in, or by a logged in user for another user account. This is controlled by the `DisplayUserName` property value. If the property is set to `true` a username text box will be displayed to the user; therefore, a user who is not logged in or is logged in using a different account can change the password based on the entered username.
- **Properties related to the current and new password:** You can retrieve the original password of the user from the `CurrentPassword` property. The new password entered by the user can be retrieved from the `NewPassword` property. Because the user is asked to enter the new password twice, the confirmation password (second entry of the new password) can be retrieved from the value of the `ConfirmNewPassword` property. You can impose restrictions on the new password and its duplicate entry, in addition to those enforced by the membership provider, by setting the `NewPasswordRegularExpression` property to a regular expression. The restrictions imposed can be displayed to the user by setting the `PasswordHintText` property. You can also configure error message text related to the new password by setting the `NewPasswordRegularExpressionErrorMessage`, `NewPasswordRequiredErrorMessage`, `ConfirmPasswordCompareErrorMessage`, and `ConfirmPasswordRequiredErrorMessage` properties. The text displayed in the label for

the current and new passwords can be customized by setting the `PasswordLabelText`, `NewPasswordLabelText`, and `ConfirmNewPasswordLabelText` properties respectively.

- **Properties related to the views:** You can set templates for the two views by using the `ChangePasswordTemplate` and `SuccessTemplate` properties. You can also configure the text to be displayed on successful change of password by setting the `SuccessText` property.
- **Properties related to Page URL:** You can define the URLs of pages that users are redirected to when they click on the Continue or Cancel buttons on the control by setting the `ContinueDestinationPageUrl` and `CancelDestinationPageUrl` properties.
- **MailDefinition:** You can use this property to configure mail settings for sending the changed password to the user in an email message.
- **MembershipProvider:** You can set this property to a membership provider.

Various other properties can be set to link the control to a page containing the `PasswordRecovery` control, `CreateUserWizard` control, or even a page that allows user profile editing.

Three important events of this class are:

- **ChangedPassword:** Raised when the password has been changed. After this event, the user is redirected to the Success page, and if mail settings are defined, the user is emailed the new password. You can handle this event to perform any processing, such as creating a custom message for the email, before the user is displayed the success page or sent an email. This event is raised by the `OnChangedPassword(EventArgs args)` method.
- **ChangePasswordError:** Raised when the password change fails due to an error. The event is raised by the `OnChangePasswordError(EventArgs args)` method.
- **ChangingPassword:** Raised just before the password change is actually done by the membership provider. You can handle this event to check the new password or process it in any desired way. The event is raised by the `OnChangingPassword(LoginCancelEventArgs args)` method.

MORE INFORMATION

This list is partial. For a complete list of events of the `PasswordRecovery` class, refer to MSDN.

TAKE NOTE *

USING THE CREATEUSERWIZARD CONTROL

The `CreateUserWizard` control is used to create new user accounts in a Web site. The `CreateUserWizard` control can also be added from the Toolbox's Login section by dragging and dropping. Optionally, you can declare it using declarative syntax or programmatically by using the `CreateUserWizard` class.

This control, and the previous two controls discussed in the lesson, function depending closely on the membership provider settings. For example, the default fields displayed in this control include the username and password. Besides these, many other optional fields, such as email address and security question/answer can be included based on the support by the membership provider.

Some of the commonly used properties of the class include `UserName`, `Password`, `PasswordHintText`, `PasswordRegularExpression`, `MailDefinition`, `MembershipProvider`, `ConfirmPassword`, `Answer`, `Question`, `ContinueDestinationPageUrl`, and so on. The usage of these properties was discussed in the controls earlier. Some more properties of this class include:

- **Email:** Defines the email address entered by the user. The `EmailRegularExpression` property can be set to impose restrictions on the email address entered.
- **RequireEmail:** Displays a field in the control where the user can enter an email address when this property is set to `true`. This property should be set to `true` to avoid an error if the membership provider requires an email address for a user account. Similarly, the `QuestionAndAnswerRequired` property should be set to `true` to display the fields for the security question/answer.
- **LoginCreatedUser:** Logs the user of the newly created account in to the site after account creation is complete when this property is set to `true`.

- **DisableCreatedUser:** Keeps created users from being immediately functional. Sometimes you may not want a newly created account to be functional instantly. For example, you may want the user to confirm the entered email address by clicking on a confirmation link mailed to him/her before the account becomes functional. In such cases, you can set this property to **true** and the **LoginCreatedUser** to **false**.

Similarly, there are various other properties of this control that can be used to customize text, error messages, styles, and titles related to page controls. Three important events of this class are **CreatedUser**, **CreatingUser**, and **CreatedUserError**. For a complete list of properties and events, refer to MSDN.

The following code example demonstrates how you can create a template **CreateUserWizard**:

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
    // How to access a control in an added wizard step
    void AddressWrite(object sender, WizardEventArgs e)
    {
        Label3.Text = "Your Address Information is:";
        Label1.Text = Server.HtmlEncode(TextBox1.Text);
        Label2.Text = Server.HtmlEncode(TextBox2.Text);
    }
    // How to access a control in a templated CreateUserStep
    void PhoneWrite(object sender, EventArgs e)
    {
        Label4.Text = "Your Phone number is:";
        CreateUserWizardStep myStep = new CreateUserWizardStep();
        Label5.Text =
        Server.HtmlEncode(((TextBox)(CreateUserWizardStep1.
        ContentTemplateContainer.FindControl("TextBoxPhone"))).Text);;
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>CreateUserWizard Templated Sample</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <h2>
                CreateUserWizard Templated Sample</h2>
            &nbsp;<asp:HyperLink ID="HyperLink1"
            NavigateUrl "~/createuserwizard.aspx" runat="server">Create a
            user</asp:HyperLink><br />
            <hr />
            <br />
            <asp:CreateUserWizard ID="CreateUserWizard1"
            runat="server" HelpPageText="Help" HelpPageUrl "~/Help.htm"
            EditProfileText="Edit Your Profile"
            EditProfileUrl "~/Profile.htm" OnFinishButtonClick="AddressWrite"
            OnCreatedUser="PhoneWrite"
            ContinueDestinationPageUrl "~/Home.htm">
                <WizardSteps>
                    <asp:CreateUserWizardStep
                    ID="CreateUserWizardStep1" runat="server">
```

```
<ContentTemplate>
    <table border="0">
        <tr>
            <td align="center" colspan="2">
                Sign Up for Your New Account</td>
        </tr>
        <tr>
            <td align="right">
                <label for="UserName">
                    User Name:</label></td>
            <td>
                <asp:TextBox ID="UserName"
runat="server"></asp:TextBox>
                <asp:RequiredFieldValidator
ControlToValidate="UserName" ErrorMessage="User Name is required."
ID="UserNameRequired"
runat="server" ToolTip="User Name is required.">
                </asp:RequiredFieldValidator>
                ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="Password">
                    Password:</label></td>
            <td>
                <asp:TextBox ID="Password"
runat="server" TextMode="Password"></asp:TextBox>
                <asp:RequiredFieldValidator
ControlToValidate="Password" ErrorMessage="Password is required."
ID="PasswordRequired"
runat="server" ToolTip="Password is required.">
                </asp:RequiredFieldValidator>
                ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="ConfirmPassword">
                    Confirm Password:</label></td>
            <td>
                <asp:TextBox ID="ConfirmPassword"
runat="server" TextMode="Password"></asp:TextBox>
                <asp:RequiredFieldValidator
ControlToValidate="ConfirmPassword" ErrorMessage="Confirm Password is
required."
ID="ConfirmPasswordRequired"
runat="server" ToolTip="Confirm Password is required.">
                </asp:RequiredFieldValidator>
                ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td align="right">
                <label for="Email">
                    E-mail:</label></td>
            <td>
                <asp:TextBox ID="Email" runat="server"></asp:TextBox>
                <asp:RequiredFieldValidator
ControlToValidate="Email" ErrorMessage="E-mail is required.">
            </td>
        </tr>
    </table>

```

```

        ID="EmailRequired" runat="server"
        ToolTip="E-mail is required."
        ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
    </td>
</tr>
<tr>
    <td align="right">
        <label for="Question">
            Security Question:</label></td>
    <td>
        <asp:TextBox ID="Question"
        runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator
        ControlToValidate="Question" ErrorMessage="Security question is
        required.">
            ID="QuestionRequired"
            runat="server" ToolTip="Security question is required."
            ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td align="right">
            <label for="Answer">
                Security Answer:</label></td>
        <td>
            <asp:TextBox ID="Answer"
            runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator
            ControlToValidate="Answer" ErrorMessage="Security answer is required."
            ID="AnswerRequired"
            runat="server" ToolTip="Security answer is required."
            ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td align="right">
            <label id="LabelPhone" runat="server" for="Phone">
                Phone:</label></td>
        <td>
            <asp:TextBox ID="TextBoxPhone"
            runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator
            ControlToValidate="TextBoxPhone" ErrorMessage="Phone number is
            required">
                ID="RequiredFieldValidator1"
            runat="server" ToolTip="Phone number is required."
            ValidationGroup="CreateUserWizard1">*</asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <td align="center" colspan="2">
            <asp:CompareValidator
            ControlToCompare="Password" ControlToValidate="ConfirmPassword"
            Display="Dynamic"
            ErrorMessage="The Password and Confirmation Password must match."
            ID="PasswordCompare" runat="server"
            ValidationGroup="CreateUserWizard1"></asp:CompareValidator>

```

```
        </td>
    </tr>
    <tr>
        <td align="center" colspan="2" style="color: red">
            <asp:Literal EnableViewState="False"
ID="ErrorMessage" runat="server"></asp:Literal>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <asp:HyperLink ID="HelpLink"
NavigateUrl "~/Help.htm" runat="server">Help</asp:HyperLink>
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <asp:TextBox ID="TextBox1"
runat="server"></asp:TextBox>
            <br />
            Address2<asp:TextBox ID="TextBox2"
runat="server"></asp:TextBox>
        </td>
    </tr>
    <asp:WizardStep>
        <asp:CompleteWizardStep
ID="CompleteWizardStep1" runat="server">
            <ContentTemplate>
                <table border="0">
                    <tr>
                        <td align="center" colspan="2">
                            Complete</td>
                    </tr>
                    <tr>
                        <td>
                            Your account has been successfully created.</td>
                        </tr>
                    <tr>
                        <td align="right" colspan="2">
                            <asp:Button CausesValidation="False"
CommandName="Continue" ID="ContinueButton" runat="server"
                                Text="Continue"
ValidationGroup="CreateUserWizard1" />
                        </td>
                    </tr>
                    <tr>
                        <td colspan="2">
                            <asp:HyperLink ID="EditProfileLink"
NavigateUrl "~/Profile.htm" runat="server">Edit
Your Profile</asp:HyperLink>
                        </td>
                    </tr>
                </table>
            </ContentTemplate>
        </asp:CompleteWizardStep>
    </asp:WizardStep>

```

```

        </td>
    </tr>
</table>
</ContentTemplate>
</asp:CompleteWizardStep>
</WizardSteps>
</asp:CreateUserWizard>
<br />
<asp:LoginName FormatString="Welcome {0}!">
    ID="LoginName1" runat="server" />
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
    &nbsp;&nbsp;<br />
    <br />
    <asp:Label ID="Label4" runat="server"></asp:Label><br />
    <br />
    <asp:Label ID="Label5" runat="server"></asp:Label><br />
    <br />
    <br />
    <asp:Label ID="Label3" runat="server"></asp:Label>
    <br />
    <br />
    <asp:Label ID="Label1" runat="server"></asp:Label><br />
    <br />
    <asp:Label ID="Label2" runat="server"></asp:Label>&nbsp;
    <br />
</div>
</form>
</body>
</html>

```

In the preceding code, note the WizardSteps tag inside the control's tag. This is used to create the steps in the user creation wizard by using the CreateUserWizardStep element. To understand the code, refer to Figures 9-8 through 9-10.

Figure 9-8

First step of wizard

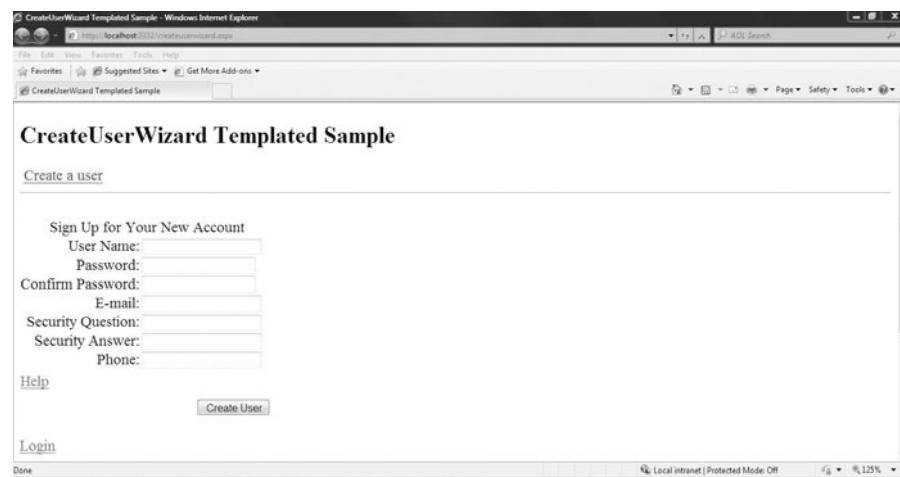


Figure 9-9 shows the second step in the wizard after the first step has been completed successfully.

Figure 9-9

Second step of wizard

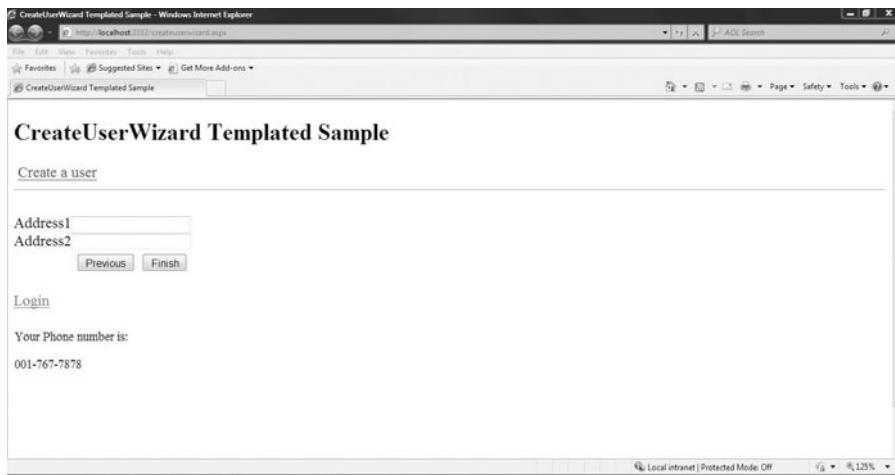
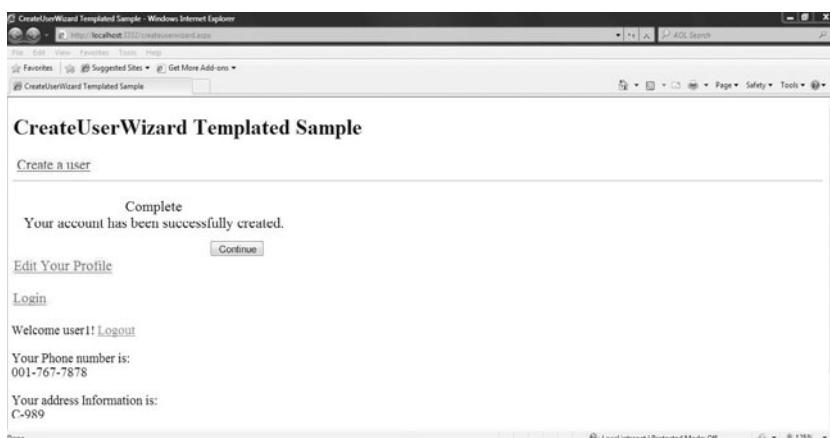


Figure 9-10 shows the completion of the wizard. Note how the user is logged on as soon as the account is created.

Figure 9-10

Last step of wizard



Implementing Authorization

Authentication validates a user's identity. After having validated the identity of the user, a secure web application should also determine the type access that each authenticated user has. For example, suppose an online news portal has two types of user memberships, free and paid, and allows paid members to view supplementary news pages and entertainment magazine pages in addition to the general news pages viewable by all users. In this case, after authenticating users, you will need to check whether they are free users or paid users and grant or deny them access to the additional pages. This is handled through authorization.

Authorization refers to determining whether an authenticated user has access to resources of the web application. In ASP.NET, you can classify authorization as File authorization and URL authorization.

File authorization checks whether the user account under which the ASP.NET engine is executing, has permissions to access a file requested through the application. The check is done based on the file system permissions configured for the user account. File authorization is based on the concept of access control list (ACL). An ACL is a list of permissions associated with a specific resource that controls which users can access that resource. For example, suppose ASP.NET engine is running under the default Windows user account ASPNET (which is unprivileged), if a request comes for a Home.aspx page from a client, file authorization checks whether the file <drive>:\<location>\home.aspx can be read by the ASPNET account. If the ACL of this file indicates that the account has the rights to access it, authorization is passed. When the authentication mode is set to Windows, file authorization is automatically implemented. The `System.Web.Security.FileAuthorizationModule` class is used to implement file authorization.

An important method of this class is the static `CheckFileAccessForUser()` method. This method accepts the virtual path of the file that you want to check access for, the Windows token representing the user account, and HTTP action methods such as GET or POST.

URL authorization controls access to parts of web applications such as directories as well as the URLs of pages in them. For example, consider the case of the news portal that allows paid users to view additional pages on the site. In this case, you can store all additional pages in a directory under the application root directory and then deny access to all free users to that directory. This is done through configuration settings in the web.config file for the specific directories. Let us discuss URL authorization in detail.

In order to implement URL authorization, you need to add the authorization element in the web.config file of a directory. The authorization element encloses the section that defines authorization-related configuration. It has the following child elements:

- **allow:** Grants access to a user or a role on the directory whose configuration file it is defined in. A role represents a set of users that needs to be assigned the same set of access privileges. For example, all paid users and all free users can be grouped into two different roles named PaidMembers and FreeUsers so that they can be denied or allowed access to a particular directory. You will learn about role management in detail later in this lesson.
- **deny:** Denies access to a user or a role on the directory whose configuration file it is defined in.

At least one of the preceding two elements is mandatory under the authorization element. The `allow` and `deny` elements have the following attributes:

- **users:** Allow/deny access on the directory to user accounts or role names by setting the value of this attribute to a comma-separated list. You can use * to indicate all authenticated users and a ? for all anonymous users (who do not have a user account associated with them).
- **roles:** Allow/deny access on the directory to user accounts or role names by setting the value of this attribute to a comma-separated list. You can use * to indicate all authenticated users and a ? for all anonymous users (who do not have a user account associated with them).
- **verbs:** Allow/deny access to HTTP action methods, such as GET, POST, or HEAD, by setting the value of this attribute to a comma-separated list. You can use * to indicate all verbs.

Let's say that you have saved all additional pages of the news portal in a directory named `supplement` under the root of the application and you want to implement the following access controls:

- Deny access to this directory to all users that belong to the FreeUsers role.
- Allow access to this directory to all users that belong to the PaidMembers role.

- Grant access to this directory to the Admin role specifically.
- Grant access to this directory to the user Manager specifically.
- Deny access to this directory to users Peter and Sam specifically because of their site policy violations.

For the preceding authorization requirements, you can add the following snippet in the web.config for the `supplement` directory under `system.web` element:

```
...
<system.web>
  <authorization>
    <allow roles="PaidMembers, Admin" users="Manager"/>
    <deny roles="FreeUsers" users="Peter, Sam"/>
  </authorization>
...
</system.web>
...
```

In the preceding code, a comma-separated list is used to allow access to roles PaidMembers and Admin. Similarly, the users Peter and Sam are denied access using a comma-separated list in the `deny` element.

The local machine user account/role names can optionally be preceded with `a.\` to indicate that they are on the local machine. Instead, if a domain user or role is being specified, the domain name as well as username must be specified as `domainName\username`. Moreover, all permissions granted/denied for a directory are applicable for its subdirectories also.

Note that you can also add separate `allow/deny` elements instead of using a comma-separated list. This is particularly useful when you want the access to granted or revoked in order of priority. For example, if you want to grant access on the directory to all users who use `POST` action, deny access to all users using the `GET` action except for the user Admin, then you should configure the settings as follows:

```
...
<system.web>
  <authorization>
    <allow users="*" verb="POST"/>
    <allow users="Admin" verb="GET"/>
    <deny users="*" verb = "GET"/>
  </authorization>
...
</system.web>
...
```

In the preceding code, note that Admin is first allowed access for `GET` action before all other users that use `GET` action are denied access. This is because first a match is found for the `Allow` element in the configuration file and if the first match is found, then the configuration is not scanned further:

```
<allow users="*" verb="POST"/>
<deny users="*" verb = "GET"/>
<allow users="Admin" verb="GET"/>
```

In this code, Admin is not granted access using `GET` because the `deny` element matches first causing all further scanning to stop.

Note that the rules defined in the `web.config` that is higher up (near the root) in the application hierarchy is given precedence over rules configured in a `web.config` that are lower in the hierarchy. In addition, if no match is found, access is granted by default because the

default application-level configuration sets `<allow users="*">`. To prevent this, you can change this to `<deny users="*">` in the application level configuration file. In this case, if a user's access privilege is not defined in the configuration files, the user will be denied access.

Maintaining and managing configuration files for each directory that you want to control access for is tedious and error-prone. To prevent this, you have an option to use the `location` element. This element can be defined in the configuration file at the application root and can be used to apply authorization settings on specific resources for subdirectories and files under the root. It has two attributes:

- **Path:** Specifies the path of a file or directory to which the configurations defined as child elements of the location element apply.
- **allowOverride:** Specifies whether the configuration files in subdirectories of the root can override the settings defined here.

To use the `location` element, place it under the root configuration element and add the configuration to apply as child elements. For example, to deny allowOverride to Peter on a specific web page, `Entertainment.aspx`, you can use the following settings:

```
<configuration>
    <location path = "Entertainment.aspx">
        <system.web>
            <authorization>
                <deny users = "Peter" allowOverride="false"/>
            </authorization>
        </system.web>
    </location>
</configuration>
```

Note how the `authorization` element is embedded in the `location` element to define authorization settings. Also, note that none of the child `web.config` files can override this setting of denying Peter access to the page.

TAKE NOTE *

The `System.Web.Security.UrlAuthorizationModule` class interfaces between the configuration file settings and the application to implement URL authorization. An important member of this class is the static method `CheckUrlAccessForPrincipal()`, which accepts the virtual path of the resource for which it has to check access privilege, the current user as an instance of a class implementing the `System.Security.Principal.IPrincipal` interface and the action method.

In the preceding topic, you learned how roles can be granted or denied access. Now, you will learn how roles can be created and managed in ASP.NET.

USING ROLE MANAGEMENT

A role is a named entity that refers to set of users that have something in common. For example, a role can be used when you want to collectively refer to a set of users that have access to a particular resource in the web application, as discussed previously. In addition, you can also use roles when you want to group users based on the tasks that they can perform in a web application. For example, consider the case of the news portal. The paid members may be allowed to perform tasks such as adding articles to a favorites menu, customizing their home page to show them articles on their own interest topics, and so on, whereas free users may not be shown the links to these operations when they log on to the Web site at all.

Role management refers to configuring your application to use role providers and creating and managing roles. A role provider is responsible for managing and storing all the data related to roles in an underlying data store. ASP.NET provides three role providers:

- **SQL Server:** Uses the SQL Server database to store role information. This is the default provider.
- **Authorization Manager or AzMan:** Uses an authorization manager XML file or a directory-based policy store.

- **Windows:** Uses Windows groups for roles and does not allow you to create roles programmatically. This is used when the authentication mode is set to Windows and is based on Windows accounts.

You can also create custom role providers by using the `System.Web.Security.RoleProvider` class.

In order to use role management in your application, you need to perform the following steps:

1. Configure the role provider.
2. Use the role management API to create and manage roles.
3. Write business logic based on role of the current user or add authorization rules in the configuration file for roles.

To configure the role provider, you can use the `roleManager` element under the `system.web` element. The `roleManager` element contains many attributes such as:

- **defaultProvider:** Defines the default role provider. In the default configuration, it is set to `AspNetSqlRoleProvider`.
- **enabled:** Defines whether roles management is enabled. In the default configuration file, it is set to `false`.
- **cacheRolesInCookie:** Defines whether the roles for the current user are cached in a cookie. When roles are cached in a cookie, and your application needs to check the role of a current user, the cookie is checked before the data store of the role provider is checked. Various other cookie related attributes can also be defined.

The child element of the `roleManager` element is `Providers`, which represents the collection of role providers and has the three child elements: `add`, `remove`, and `clear`. You can add settings other than the default settings by using the `add` element. For example, if you want to add a `MyRoleProvider` for your `MyApp` application that uses the SQL Server store, you can do as follows:

```
<roleManager defaultProvider="MyRoleProvider"  
enabled="true" cacheRolesInCookie="true">  
    <providers>  
        <add name="MyRoleProvider"  
            connectionStringName="myConStr" applicationName="MyApp"  
            type="System.Web.Security.SqlRoleProvider, System.Web, Version=2.0.0.0,  
            Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />  
        <add .../>  
    ...  
    </providers>  
</roleManager>
```

MORE INFORMATION

For a complete list of attributes of the `add` element, refer to MSDN.

After configuring the application, you should be able to create roles and manage them. This can be done programmatically through the role management API provided by ASP.NET of which the `Roles` class is primary.

The `Roles` class exposes various static properties and methods that correspond to the attributes of the `roleManager` element in the configuration file. Some of the commonly used properties are `ApplicationName`, `Enabled`, `CacheRolesInCookie`, and `Provider` (retrieves the default provider). You can use these properties to retrieve the values defined in the configuration file.

The commonly used methods of the `Roles` class are:

- **CreateRole():** You can use this method to create a new role that passes the name as the string argument to the method. The role is created and added to the roles in the underlying data source through the role provider.
- **AddUserToRole():** You can use this method to add a user to a role as its member. This method accepts the username and the name of the role as string arguments.

You can also add a single user to multiple roles by using the `AddUserToRoles()` method, which accepts the second argument as an array of string values depicting role names to which the username has to be added. Similarly, you can add multiple users to a single role by using the `AddUsersToRole()` method that accepts an array of strings as the first argument for all usernames to be added. In addition, you can add multiple users to multiple roles by using the `AddUsersToRoles()` method, which accepts string arrays for both arguments. Similar to the Add series of method, you have the Remove methods, such as `RemoveUserFromRole()` and `RemoveUsersFromRoles()` that remove one or more users from one or more roles.

- **DeleteRole():** You can use this overloaded method to delete a role from the data store of the current role provider. One overloaded form only accepts the name of the role to be deleted as a string argument. The second overloaded form also accepts a second argument that can be set to `true` to throw an exception if users (members) of the role being deleted exist. The overloaded forms return `true` or `false` depending on whether the role was successfully deleted.
- **Methods related to retrieving roles and users in roles:** You can use the `GetAllRoles()` method to retrieve all role names stored in the data store for the application defined as the value of the `applicationName` attribute in the configuration file. The list of role names is returned as string array. You can also retrieve the roles for a specific user by using the `GetRolesForUser()` method. This method accepts the username as an argument and returns a string array for the list of roles to which the user belongs. Similarly, you can retrieve all users in a role by using the `GetUsersInRole()` method, which accepts the role name as an argument and returns the list of users in the role as a string array. You can also find a specific user in a specific role by using the `FindUserInRole()` method, which accepts the role name and username as the arguments.
- **RoleExists():** You can use this method to check whether the role whose name is passed an argument exists in the data store.
- **IsUserInRole():** You can use this overloaded method to check whether the current or a specific user is in a role. One overloaded form accepts a role name and checks for the current user in that role. The second overloaded form accepts a username and a role name as arguments and checks for the user in the specified role. The method returns `true` or `false` based on the results of the check.
- **DeleteCookie():** You can use this method to delete the cookie that caches the role names.

TAKE NOTE*

You can also use the Web Site Administration Tool to create and manage roles.

To check at runtime whether the currently logged in user is in a role, you can use the `IsInRole()` method of the `User` object and pass the name of the role to the method. Optionally, you can use the `RolePrincipal` class instance added to the current request context and invoke its methods.

USING IMPERSONATION

Sometimes you may not want to handle the authentication and authorization functionality in your code. In that case, if you want to base your authentication and authorization functionality on IIS, then you can use impersonation. Impersonation refers to the technique of allowing a process to run under the user identity of another process. In this case, your application code executes under the Windows identity of the client requesting the code execution.

When you use impersonation, compilation and configuration file loading are not done using the impersonated identity. Instead they are done using the identity of the ASP.NET engine process. Impersonation is disabled by default. When you enable impersonation, IIS passes the access token representing the identity of the client to ASP.NET and runs the code of the application requested by that client under the passed identity. Note that the identity passed can be of an authenticated user or an anonymous user.

To configure client impersonation, you use the **identity** element under the **system.web** section in the configuration file. The element has the following three attributes that facilitate impersonation:

- **Impersonate:** Set this mandatory attribute to **true** to use client impersonation for the application. To disable the use of client impersonation, set this attribute to **false**. When impersonation is enabled, and anonymous access is enabled in IIS, the **IUSR_machinename** account is used to send the request to the application. However, if impersonation is enabled, but anonymous access is disabled in IIS, the authenticated identity is used for the request. Note that regardless of the identity, the access privileges are controlled per those configured on the NTFS file system. When impersonation is disabled but anonymous access is enabled in IIS, the process user account is used for the request.
- **userName:** Set the username in this optional attribute when you want the impersonation to be done for a specific user account other than the requesting client.
- **password:** Set the password in this optional attribute when you want the impersonation to be done for a specific user account other than the requesting client.

TAKE NOTE*

The username and password can be set as clear text, or for security reasons, they can be set as encrypted values. For details, refer to MSDN.

You can use the **System.Security.Principal.WindowsIdentity** class to retrieve the identity of the user under which the application is running even during impersonation. This can be done using **System.Security.Principal.WindowsIdentity.GetCurrent().Name**.

The following code shows an example of enabling impersonation for every client from which requests originate:

```
<configuration>
  <system.web>
    ...
    <identity impersonate="true"/>
  ...
</system.web>
</configuration>
```

The following code shows an example of enabling impersonation for a specific user:

```
<configuration>
  <system.web>
    ...
    <identity impersonate="true"
      userName="www1\Peter" password="pwd1" />
  ...
</system.web>
</configuration>
```

In the preceding snippet, www1 is the domain name, Peter is the username, and pwd1 is the password. When you use such impersonation, you can connect to a computer on the network and request resources from it; for example, when your SQL Server database is on a SQL Server instance running on another computer. If you do not specify the credentials in the configuration and are using Basic authentication in IIS, then you can connect to the network and request resources from another computer.

TAKE NOTE*

You must set the username and password when you want to request resources from a computer on behalf of the user that the application is impersonating if the **autoConfig** attribute of the **processModel** element is set to **false** in the configuration file. If this attribute is set to **true**, impersonation is automatically handled by ASP.NET.

You can also do impersonation programmatically for a particular section of code by using the **Impersonate()** method of the **WindowsIdentity** class. The method returns an instance of

CERTIFICATION READY?

Configure authentication, authorization, and impersonation.

1.2

TAKE NOTE*

Another property of the `WindowsIdentity` class relevant to impersonation is `ImpersonationLevel`. For information on this property, refer to MSDN.

USE THE OUTPUTCACHE DIRECTIVE

To use the `OutputCache` directive:

1. Add a new web page in your web application.
2. Add a Label control, a drop-down list control, and a Button control.
3. Add five items in the collection of the drop-down list control, as shown in this markup:

```
<form id="form1" runat="server">
    <asp:Label ID="Label1" runat="server"></asp:Label>
    <br />
    <br />
    Select an Item:
    <asp:DropDownList ID="DropDownList1" runat="server" >
        <asp:ListItem>Item 1</asp:ListItem>
        <asp:ListItem>Item 2</asp:ListItem>
        <asp:ListItem>Item 3</asp:ListItem>
        <asp:ListItem>Item 4</asp:ListItem>
        <asp:ListItem>Item 5</asp:ListItem>
    </asp:DropDownList>
    <br />
    <asp:Button ID="btnSubmit" runat="server"
    onclick="btnSubmit_Click" Text="Submit" />
```

4. In the `Click` event handler of the Button control, add the following code that displays the selected item message on the label:

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    Label1.Text = "You've selected " +
    DropDownList1.Text + " at " + DateTime.Now.ToString();
}
```

5. Run the program. Note that when you select an item from the drop-down list and click the Submit button, it displays a message with the selected item's text and the time stamp.
 6. Now, open the markup for the web page and add the following line below the `<@Page>` directive to cache the page for five seconds:
- ```
<%@ OutputCache Duration="5" VaryByParam="None" %>
```
7. Now, run the program. Select different values from the drop-down list, and click the Submit button. Note that the page displays the cached data even when you

change the selected item from the drop-down list. After five seconds, the page will display the correct message based on the selected item.

8. Again go back to the markup of the page and change the OutputCache directive, as shown:

```
<%@ OutputCache Duration="10" VaryByParam="DropDownList1" %>
```

9. Run the program. Select Item 2, and click the Submit button. Then select Item 1, and click the Submit button. Again select Item 2. Note that when you select Item 2 the second time, you get a cached page with the old time stamp.

## SKILL SUMMARY

In this lesson, you learned about improving the performance of web applications by using caching. You also learned how to add data to the cache and retrieve it. To implement caching, it is important to define cache dependency and expiration. Caching can be of two types: application data caching and page output caching. Page output cache can be used to cache the page and you can declare it using the @OutputCache directive. You saw how to programmatically configure caching for a single page and for multiple pages.

This lesson also dealt with security of web applications. You learned how to secure web applications using authentication, authorization, and membership providers. The lesson also discussed the different login controls provided by ASP.NET such as Login, CreateUserWizard, and LoginStatus. Next, it covered how to use the membership providers, such as ActiveDirectoryMembershipProvider and SQLMembershipProvider for authentication and how to configure authentication and authorization in the web.config file. Finally, you learned about roles, user identity, and impersonation and how you can use these to configure access for users of your application.

For the certification examination:

- Implement and manage application data caching to improve performance.
- Understand and implement providers.
- Use security controls to configure Windows, Forms, Passport, and Anonymous Authentication.

## ■ Knowledge Assessment

### Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. The Cache object is available as a property of the \_\_\_\_\_ class.
2. \_\_\_\_\_ are the modules that actually contain all the code to implement the membership feature.
3. The \_\_\_\_\_ object allows you to make runtime decisions about output caching.
4. \_\_\_\_\_ is the process of validating the user credentials against the membership providers such as Active Directory or SQL Server.
5. \_\_\_\_\_ refers to the process of determining whether an identity has the rights to access a specific resource.

### True / False

---

*Circle T if the statement is true or F if the statement is false.*

- |          |                                                                                                                      |
|----------|----------------------------------------------------------------------------------------------------------------------|
| <b>T</b> | <b>F</b> 1. The LoginView control allows display of information based on the user.                                   |
| <b>T</b> | <b>F</b> 2. The LoginStatus control is used to display the login link for users who are not authenticated.           |
| <b>T</b> | <b>F</b> 3. You can use the Add() method to insert a new item in the cache or update an existing item in the cache.  |
| <b>T</b> | <b>F</b> 4. You can define multiple dependencies for a single cached item using the AggregateCacheDependency object. |
| <b>T</b> | <b>F</b> 5. You can make runtime decisions about output caching using the Response.Cache object.                     |

### Multiple Choice

---

*Circle the letter or letters that correspond to the best answer or answers.*

1. When you want to cache data for a specific amount of time, which parameter do you need to pass to the Cache.Insert method?
  - a. slidingExpiration
  - b. cookieExpiration
  - c. absoluteExpiration
  - d. cacheExpiration
2. Which of the following controls is used to add new users to the ASP.NET membership system?
  - a. LoginStatus
  - b. ChangePassword
  - c. CreateUserWizard
  - d. LoginView
3. The Cache class simplifies the use and implementation of application data caching by providing access to data stored in cache in the form of \_\_\_\_\_?
  - a. Binary data
  - b. Integers
  - c. key-value pairs
  - d. Strings
4. The Login controls use the methods of \_\_\_\_\_ internally?
  - a. The System.Web.Roles class
  - b. The System.Web.Membership class
  - c. The System.Security.Membership.Web class
  - d. The System.Web.Security.Membership class
5. Which authorization can you use to allow or deny access to a specific directory by user-name or role?
  - a. File authorization
  - b. URL authorization
  - c. Folder authorization
  - d. Windows authorization

### Review Questions

---

1. Why do you use impersonation?
2. What is Passport authentication used for?

## ■ Case Scenarios

### **Scenario 9-1: Improving Web Application Performance**

You are developing a web application to record employee attendance. This involves inserting details into the database and displaying the details to the employees. Additionally, you want to get the attendance report for the specified month. How will you improve the performance of the web application?

### **Scenario 9-2: Authenticating and Authorizing Users**

You've developed an employee attendance application. How will you ensure that only legitimate users can access it? Additionally, you need to ensure that the user of the system should only access the details that are permissible to that user.



## **Workplace Ready**

### **Performance Improvement**

In today's world, software applications play an important role in most business solutions. Software applications are based on events. With proper user input, these applications can perform appropriate actions and then display the results to users. Application performance is an important aspect. For example, suppose you are creating a Web site that allows users to play games online. In this application, quick user response is an important aspect. How will you ensure better application performance?

# Working with Services, Windows Communication Foundation, and ASP.NET Extensions

## OBJECTIVE DOMAIN MATRIX

| TECHNOLOGY SKILL                                    | OBJECTIVE DOMAIN                                                                                 | OBJECTIVE DOMAIN NUMBER |
|-----------------------------------------------------|--------------------------------------------------------------------------------------------------|-------------------------|
| Introducing ASP.NET Web Services.                   | Consume services from client scripts.                                                            | 5.3                     |
| Introducing ASP.NET Web Services.                   | Read and write XML data.                                                                         | 3.1                     |
| Understanding the Windows Communication Foundation. | Call a Windows Communication Foundation (WCF) service or a web service from an ASP.NET web page. | 3.3                     |
| Understanding ASP.NET Extensions.                   | N/A                                                                                              | NA                      |

## KEY TERMS

**Model View Controller (MVC)**

**Service-Oriented Architecture (SOA)**

**Simple Object Access Protocol (SOAP)**

**Web Services Description Language (WSDL)**

**Windows Communication Foundation (WCF)**

**XML Web Service**

ASP.NET 3.5 provides some new features that you can use to develop datacentric applications with ease. You can create web services and reuse these within and across applications. You can reuse controls using Dynamic Data. You can create rich Flash-like applications using Silverlight. The possibilities are endless.

## ■ Introducing ASP.NET Web Services



**THE BOTTOM LINE**

A web service is a software system that enables interoperability of applications across domains and networks. Every web service has an interface in a specific format known as **Web Services Description Language (WSDL)**, which specifies the target Uniform Resource Locator (URL) of a service and the format in which methods should be wrapped and messages should be encoded in the service. Other systems communicate with XML Web Services using **Simple Object Access Protocol (SOAP)** messages with XML serialization and HTTP as the underlying standard.

The programming model provided by ASP.NET allows you to use Internet and standard protocols by wrapping the application code as a web service object.

## Creating and Using ASP.NET Web Services

---

You use web services in ASP.NET applications to enable the wider use of code across applications, domains, services, and networks. These web services use HTTP and XML for communication and are called **XML Web Services**. You can also create web services using the **Windows Communication Foundation (WCF)**.

Like web pages, web services are exposed using URLs. You can call the web service in any of your .NET applications using a proxy object. This proxy object is automatically created when you add a reference to the service. The proxy object is responsible for serializing the message and sending it to the web service.

A **WebService** is a class that you derive from the **System.Web.Services.WebService** class provided by the .NET Framework. This is a full-functionality class that implements the core capabilities needed for web services functioning. This class processes the service requests and provides access to ASP.NET features like Application and Session similar to the ASP.NET web applications. The web service is typically an .asmx file. You need to instantiate the **WebService** class to create a web service object. You can use this object to deserialize requests, execute code, and respond to the application based on the request.

Let's now discuss the classes that you can use in your web services.

### USING CLASSES IN WEB SERVICES

ASP.NET provides some classes that you can use to add specific functionality to your web services. Let's learn about these.

#### Using the **WebServiceAttribute** Class

The **WebServiceAttribute** class allows you to provide additional functionality to the **WebService** class. You can use the following parameters of the **WebServiceAttribute** class to provide additional information about the web services to client applications:

- **Namespace:** Sets the namespace for the web service. This namespace should be a domain that is in your control.
- **Description:** Contains text that identifies the purpose of the web service.
- **Name:** Contains text that identifies the name of the web service.

To add the **WebServiceAttribute** class in your web service, use the following syntax:

```
[WebService(Namespace = "YourNamespace")]
public class YourService: System.Web.Services.WebService
```

#### Using the **WebService** Class

You use the **WebService** class to access ASP.NET objects, such as **Application** and **Session**. The **WebService** class is the base class for web service applications in ASP.NET. To use this class, you need to first inherit it. Then you can access the **Session** object, just like you would from an ASP.NET web page. This class also handles working with various web service standards, such as SOAP and XML.

Here are the important properties of this class:

- **Application:** Gets the application object for the current HTTP request.
- **Context:** Gets the ASP.NET **System.Web.HttpContext** for the current request.
- **Session:** Gets the **HttpSessionState** instance for the current request.

- **SoapVersion:** Gets the version of the Simple Object Access Protocol (SOAP) used to make the SOAP request to the XML Web Service.
- **User:** Gets the ASP.NET server System.Web.HttpContext.User object. Can be used to authenticate whether a user is authorized to execute the request or not.

### Using the WebMethodAttribute Class

You need to use the `WebMethodAttribute` class for the methods that you want to expose. You can apply this attribute to the web service without defining any parameters to identify the methods. If you apply the attribute on the service, you don't have to individually apply it on each method.

The `WebMethodAttribute` class has the following parameters:

- **EnableSessionState:** Indicates whether the method will work with the session state.
- **MessageName:** Indicates the name used for the web service method.
- **TransactionOption:** Indicates whether the method supports transactions.
- **Description:** Specifies a description for the web method.
- **CacheDuration:** Defines the number of seconds for the server to cache the response.
- **BufferResponse:** Indicates whether the web service response to the client needs to be buffered.

The following code snippet shows how you can apply this attribute:

```
[WebService(Namespace = "Service1")]
public class Service1: System.Web.Services.WebService
{
 [WebMethod]
 public string HelloWorld()
 {
 return "Hello World!";
 }
}
```

### Using the ScriptServiceAttribute Class

The `ScriptServiceAttribute` class is used to indicate that the web service can be called from script. This is a sealed class that doesn't have any properties, methods, or fields. This class has only a default construct that initializes the new instance of the `ScriptServiceAttribute` class.

Now, let us see how to create a simple web service in ASP.NET.

## CREATING AND USING XML WEB SERVICES

Let's create a simple web service that will return the string "Hello World." To do this, first you need to create the web service, and then you need to create a web application that will display the string Hello World by calling that web service.



### CREATE AN XML WEB SERVICE

---

To create a new web service project:

1. Select File > New Project.
2. From the New Project dialog box, select ASP.NET Web Service Application.
3. Specify the name of the project, say, MyWebService. This creates a project that is similar in structure to your Web site. Because this project is hosted within ASP.NET, all ASP.NET features, such as session, security model, and configuration are available to this project. Here is the markup of the `.asmx` file that gets created for the web service:

```
<%@ WebService Language="C#"
CodeBehind="Service1.asmx.cs"
Class="MyWebService.Service1" %>
```

A class named Service1 is generated. Visual Studio also generates the code as shown:

```
public class Service1: System.Web.Services.WebService
{
 [WebMethod]
 public string HelloWorld()
 {
 return "Hello World";
 }
}
```

#### CERTIFICATION READY?

Read and write XML data.  
3.1

You can add the web service file (defined by the .asmx extension) to an existing Web site. Alternatively, you can create a web application exclusively for web services and browse through them using the URL.

## USING ASP.NET WEB SERVICES

You can use ASP.NET XML Web Services from any application that can use HTTP protocol for communication. You can also browse your web service in the web browser by specifying the URL of the web service like `http://localhost/WebService1/Service1.asmx`. This shows the description of the web service in the web browser.

To use ASP.NET Web Service, you first reference the web service and then call it. In addition, you need to ensure the security of the web service.

To get started, you must have a published web service and subscribe to it. Once you have a published web service, you set up a reference to that web service.



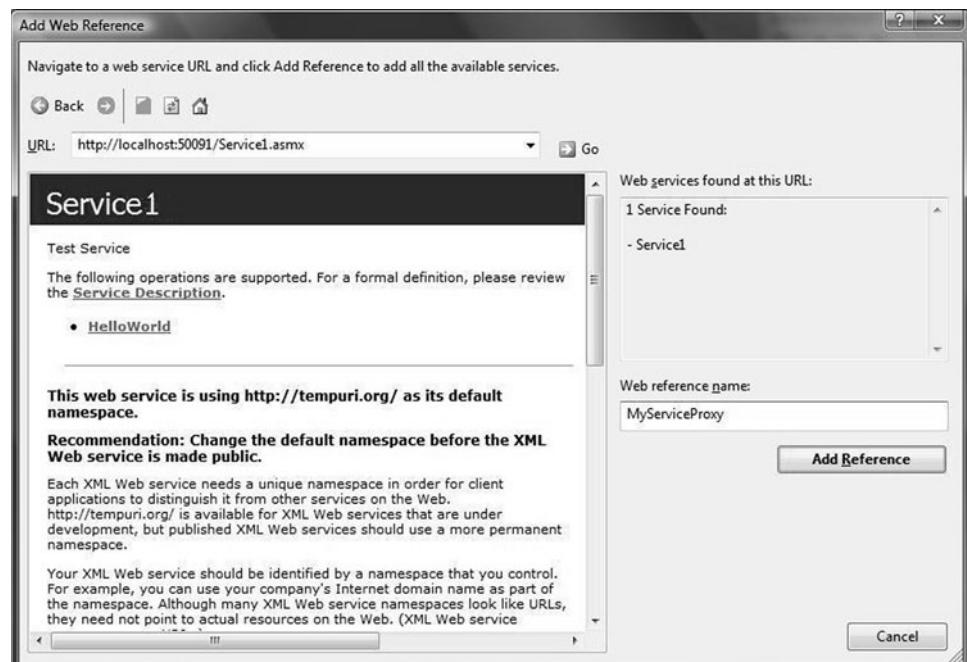
## REFERENCE A WEB SERVICE

Let's use the web service created earlier in the MyWebService project. To do this:

1. Right click the Web Application project where you want to call the service, and select Add Web Reference. This opens the Add Web Reference dialog box as shown in Figure 10-1.

**Figure 10-1**

The Add Web Reference dialog box



2. Provide the URL of the web service.
3. Select the web service, and enter a name for this reference, for example <http://localhost/WebApplication1/Service1.asmx>. A proxy class is automatically generated with the name of this reference. The generated proxy class will use this name to define the namespace for accessing the web service.

After performing these steps, Visual Studio will generate a proxy class for this reference that will handle all web service communication, serialization, and deserialization of requests and responses. The proxy class is required to communicate to and from the web service.

When you right click on the project, you will get the Add Web Reference and Add Service Reference options. Ensure that you select the Add Web Reference option for XML Web Service. The Add Service Reference option is used to call the WCF service.

After referencing the web service, you need to call it by making use of the proxy class. This is similar to calling a method from a class. You can bind to the web service call through the local proxy class and retrieve information that is displayed in the GridView or FormView controls. The following code snippet shows you how to call the web service:

```
MyServiceProxy. Service1 service = new MyServiceProxy. Service1();
Response.Write(service.HelloWorld());
```

Let's see an example. Say, you have created an XML Web Service StudentASMX that returns a list of students in a university. Now you need to create a web application and call the web service to display the list of students in GridView.



## CALL A WEB SERVICE

---

Now that you've created the web service, let's create a web application to call the web service. To do this:

1. Add a new ASP.NET web application with name Student Application in the solution.
  2. Go to the markup of the default.aspx page and add a new GridView control, as shown in this markup:
- ```
<form id="form1" runat="server">
<asp:GridView ID="GridView1" runat="server" />
</form>
```
3. Right click the ASP.NET application, and select the Add Web Reference option. This will open the Add Web Reference dialog box.
 4. Select Web Services in the This Solution option, type the name StudentServiceProxy in the web reference name, and click the Add Reference button.
 5. Go to the code behind of the default.aspx and, in the Page_Load method, call the web service method GetStudentList, and bind the result to the GridView control as shown here:

```
protected void Page_Load(object sender, EventArgs e)
{
    StudentServiceProxy.StudentService service = new
    StudentServiceProxy.StudentService();
    GridView1.DataSource = service.GetStudentList();
    GridView1.DataBind();
}
```

CREATING AN AJAX WEB SERVICE

Alternatively, you can call the web service directly from the client-side JavaScript through AJAX. To do this, the client-side page and the XML Web Service should be in the same domain.



CREATE AN AJAX WEB SERVICE

To call the web service from the client-side JavaScript:

1. Indicate that the methods can be called from the client script. To do this, mark the web service with `ScriptServiceAttribute` as shown:

```
[System.Web.Script.Services.ScriptService]
```

2. Register the HTTP `ScriptHandlerFactory` with the Web site in the `web.config` file as shown:

```
<httpHandlers>
<remove verb="*" path="*.asmx"/>
<add verb="*" path="*.asmx" validate="false" type="System.Web.
Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
<add verb="*" path="*_AppService.axd" validate="false" type="System.
Web.Script.Services.ScriptHandlerFactory, System.Web.Extensions,
Version=3.5.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
<add verb="GET,HEAD" path="ScriptResource.axd"
type="System.Web.Handlers.ScriptResourceHandler,
System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" validate="false"/>
</httpHandlers>
```

3. Add a `ScriptManager` to the client page that references the XML Web Service as shown in the code snippet:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="http://localhost/WebService1/
Service1.asmx" />
  </Services>
</asp:ScriptManager>
```

4. Add another JavaScript method to the client page that will act as the callback handler. This will receive the results of the web service handler:

```
<script type="text/javascript" >
function CallService()
{
  WebService1.HelloWorld(GetResult);
}
function GetResult(result)
{
  // Result variable holds the result of the method call
}
</script>
```

5. Add a new button to the page:

```
<input id="Button1" type="button" value="Call"
onclick="CallService()" />
```

When you click the button, the web service is executed.



Refer to Lesson 6
“Working with ASP.NET
AJAX” for more information on AJAX.

USING SOAP HEADERS TO SECURE WEB SERVICES

It is necessary to secure the web services that you create for your application. But before deciding on the kind of security that you wish to implement, you need to weigh your performance requirements against your security requirements.

Using SOAP headers is a common way to secure web services.

XML Web Services use SOAP as an underlying protocol. You can define these SOAP headers for your web service. Use this method when the calling clients cannot participate in the standard,

Windows-based security model. For web services that require interoperability, you can use SOAP headers to pass user credentials. Unlike the ASP.NET security model, this standard is based on the web services standard.

To define your own SOAP header for the web service, you should create a class that will inherit the `SoapHeader` class:

```
public class TestSoapHeader: SoapHeader
```

Next, you should add properties corresponding to each element in the SOAP header:

```
public class TestSoapHeader: SoapHeader
{
    public DateTime Created;
    public long Expires;
}
```

Once you've created the `SoapHeader`, the next step is to process the `SoapHeader` in the web service. Create a new web service called `TestWebService`. Go to the code behind of the web service and add a public member of type `TestSoapHeader`. Apply the `SoapHeader` attribute to the `HelloWorld`:

```
[WebService]
public class TestWebService:
System.Web.Services.WebService
{
    public TestSoapHeader time;

    [SoapHeader("time", Direction=SoapHeaderDirection.InOut)]
    [WebMethod]
    public string HelloWorld()
    {
        if (time == null)
            time = new TestSoapHeader();
        time.Created = DateTime.Now;
        time.Expires = 10000;
        return "Hello World";
    }
}
```

To view the SOAP response generated by the web service, you can view the web service in a web browser and click on the `HelloWorld` method. This will show the SOAP response as shown:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Header>
        <TestSoapHeader xmlns="http://tempuri.org/">
            <Created>dateTime</Created>
            <Expires>long</Expires>
        </TestSoapHeader>
    </soap:Header>
    <soap:Body>
        <HelloWorld xmlns="http://tempuri.org/" />
    </soap:Body>
</soap:Envelope>
```

This is how you can create a SOAP header for your web service. You can also implement security for using the SOAP headers by customizing the SOAP headers to be used in a secure and encrypted manner or in plain text based on your needs. SOAP headers do not have any

default or built-in features for authentication. You need to specify how the client and service need to process the header information.

MORE INFORMATION

For more information on implementing security with SOAP headers, refer to the "Perform Custom Authentication Using SOAP Headers" section on MSDN.

USING AUTHORIZATION AND AUTHENTICATION TO SECURE WEB SERVICES

In addition to implementing security using SOAP headers, you can use data encryption or authentication and authorization to restrict data access for users.

Authentication refers to validating the identity of a user accessing a web application. However, after having validated the user's identity, a secure web application should also determine the type access that each authenticated user has. Authorization refers to determining whether an authenticated user has access to resources of the web application.

ASP.NET provides several different authorization authentication methods. Let's discuss them.

In ASP.NET, you can classify authorization as File authorization and URL authorization. Using these authorization methods, you can restrict the use of web services to only authorized users. File authorization checks whether the user account under which the ASP.NET engine is executing has permissions to access a file requested through the application. The drawback of this approach is that the transmission of the credentials, logon ID, and password is in clear text. Because these credentials are not encrypted, they can be intercepted easily, making the security susceptible to network monitoring tools. URL authorization controls access to parts of a web application such as directories and the page URLs contained in them.

There are also different authentication methods. You can authenticate users by using Windows Digest, which converts passwords in hashed format. As a result, the passwords are difficult to decrypt. To use this type of security, you do not need Secure Socket Layer (SSL). The drawback of using this method is that other platforms do not support this model, which poses interoperability problems.

Another authentication method in ASP.NET is encrypting authentication credentials over SSL. This adds to the security by encrypting logon and password information. However, because it takes time to encrypt and decrypt text, it reduces performance.

You can also secure web services by using Windows Integrated security for authentication. Windows Integrated security securely passes the encrypted credentials to the server. However, both the client and the server need to run on Windows to use this security model, and you are limited to using Internet Explorer.



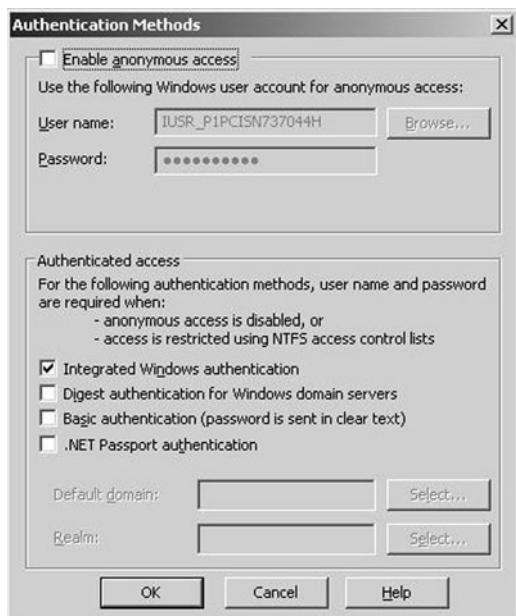
CONFIGURE SECURITY

To configure security of ASP.NET Web services:

1. Start IIS Manager.
2. Expand localhost, and then, expand Web Sites.
3. In the console tree, right click the Web site or virtual directory for which you want to configure authentication.
4. Click Properties.
5. Click the Directory Security tab, and then, under Anonymous and access control, click Edit.
6. Click to select the check box next to the authentication method that you want to use, and then, click OK. You can choose from the authentication methods as shown in Figure 10-2.

Figure 10-2

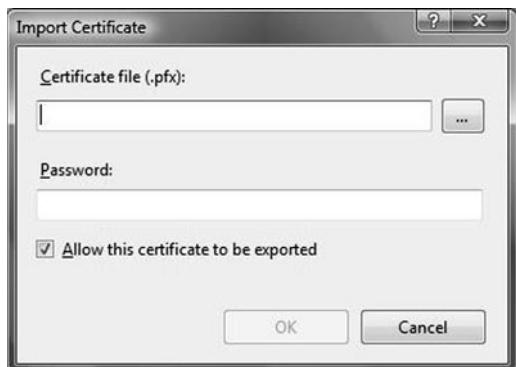
Authentication methods



You can also secure your web service by providing client certificates. You need to obtain client certificates from a trusted third-party certificate authority. A certificate is used to identify both the client and the web service when you use SSL. To import the client certificates, open IIS Manager, and double click the Server Certificate option, this will open the Server Certificates. Right click and select Import option. Now you can browse and locate the certificate file. Additionally, you need to specify the password for this certificate file. Select the Allow this certificate to be exported option if you want to export this certificate. Figure 10-3 shows the Import Certificate dialog box.

Figure 10-3

Import Certificate dialog box

**TAKE NOTE ***

You need to decide which method to use for authentication based on your performance and security requirements. Your choice of security will depend on the sensitivity and criticality of data that is being transmitted. Authentication can, however, reduce the performance of your web service. If the data being transmitted is generic, such as the score card of a game, then authorizing the user is sufficient and you need not use encryption of data. This will improve the performance of your web service and enable the service to be more responsive to the user.

CERTIFICATION READY?

Consume services from client scripts.

■ Understanding the Windows Communication Foundation

 **THE BOTTOM LINE**

The growth and widespread use of web services has impacted software development, especially in the standard protocols used for communication between applications. These days, web applications are designed to provide security, distributed transaction coordination, interoperability, and communication. The software used to develop these web applications easily allows the benefits to be passed to end users. Microsoft **Windows Communication Foundation (WCF)** provides several features that enable you to provide these benefits in your web applications.

The introduction of XML Web Services in ASP.NET has increased the use of distributed programming and led to a number of distributed communication models.

WCF is an application programming interface (API) in the .NET Framework that helps you build connected, service-oriented applications. WCF provides a means to achieve distributed computing where multiple computers running on different operating systems connect with each other using a common communication protocol.

This section discusses the importance of WCF in ASP.NET development and presents an overview of using WCF to create a web service.

Understanding the Need for WCF

WCF unifies multiple Windows communication APIs. It provides a model in which you can use previously used, distributed communication models for Windows-based applications.

Earlier, when you had to use elements, such as message queues and .NET Remoting with web services, it was cumbersome because you had to write the code and also call and host it multiple times. The WCF programming model helps you to use all these elements together. You can create a single service to use all these elements in your code.

WCF meets all **Service-Oriented Architecture (SOA)** guidelines and provides both high performance and interoperability. It can communicate efficiently through XML Web Services as well as custom XML formats and Really Simple Syndication (RSS). It provides a layer for communication in applications. This layer is independent and allows you to easily configure and distribute an application using WCF. Using an independent layer also allows you to easily address issues related to security, reliability, concurrency, transactions, serialization, error handling, and instance management.

WCF can be used with multiple protocols, as opposed to web services, which can only be used with HTTP. In addition, WCF uses SOAP-based XML data and message contracts, which improves interoperability. Improved serialization techniques have improved the performance of WCF over web services. All these features help WCF deliver superior performance.

 **MORE INFORMATION**

For a detailed comparison of WCF with other distributed communication technologies, such as web services, refer to "A Performance Comparison of Windows Communication Foundation (WCF) with Existing Distributed Communication Technologies" on MSDN. You can also refer to infosysblogs.com for a brief on the reasons for WCF's high performance.

Identifying WCF Elements

As you saw in the previous section, WCF provides an independent layer to configure and distribute an application. You can break up this layer into sublayers based on the functions that each sublayer performs.

Typically, the functions that a WCF service performs are:

- Defining contracts
- Controlling service execution at runtime
- Processing messages
- Hosting the services

In order to perform these functions and provide communication, WCF comes as a complete package. Let's now see the elements of this package.

Using WCF, you can send messages from the point of origin to the point of termination. These points are known as endpoints. Each endpoint has three elements:

- **Address:** Identifies the location of the endpoint on the network. WCF supports fully qualified addresses as well as relative addresses.
- **Binding:** Specifies how to transmit the message. WCF binding elements include a protocol, a transport, and an encoder. The binding should at least specify the transport, such as HTTP and MSMQ.
- **Contract:** Specifies the information that can be exchanged during a service call, such as namespace of the service and callbacks.

Each message consists of an envelope, a header, a body, the addressing information, and the information being exchanged. The message exchange patterns supported by WCF are:

- **One-way messages:** Pass a message from the sender to the receiver only.
- **Request-response messages:** Pass a message using the request-response pattern. The receiver is expected to reply back to the sender. This is the default message exchange pattern.
- **Duplex messages:** Pass a response back to the sender while executing a service requested by it.

In addition to endpoints and messages, WCF is also made up of the following elements:

- **Channels:** Represent how the message will be transmitted. WCF uses protocol channels to add services and transport channels to manage physical movement of data between endpoints.
- **Behaviors:** Defined by the service contract in WCF. These behaviors need not necessarily be limited to the contract. The behaviors are governed by the `ServiceBehaviorAttribute` and the `OperationBehaviorAttribute` that control the service execution.

Creating and Using a WCF Service

There are many options for creating, configuring, and hosting WCF services. The typical steps for creating and using WCF services include defining the contract, writing the service, specifying the endpoints, hosting the service, and finally calling the service from an application.

In this section, you will learn to create a WCF service and use it from an application. Let's create a WCF service that returns a list of all the students in a university.



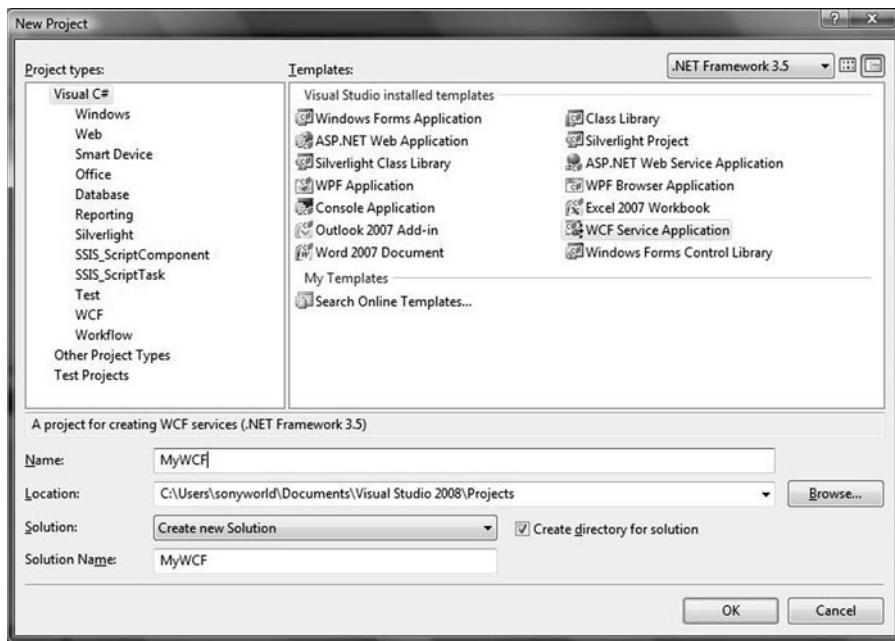
CREATE A WCF SERVICE

To create the WCF service:

1. Start Visual Studio 2008, and select File > New > Project.
2. Choose WCF Service Application from the available templates, name the application MyWCF as shown in Figure 10-4, and click OK.

Figure 10-4

The New Project dialog box

**TAKE NOTE ***

At the root of the application, there are three files by default: `IService1.cs`, `Service1.svc`, and `Service1.svc.cs`. These files are placeholders representing the WCF contract and a class implementing the contract.

3. Rename the code files representing the service. Rename `IService.cs` as `IStudent.cs`, `Service1.svc` as `StudentService.svc`, and `Service1.svc.cs` as `StudentService.svc.cs`. Ensure that you rename the service name in the `.svc` file's `<@ServiceHost>` tag. The markup for this service should look like this:

```
<%@ ServiceHost Language="C#" Debug="true"
Service="MyWCF.StudentService"
CodeBehind="StudentService.svc.cs" %>
```

4. Change the service interface name from `IService` to `IStudent` in the `IStudent.cs` file, and change the service class name from `Service` to `StudentService` in the `StudentService.svc.cs` file.
5. In the `IStudent.cs` file, update the interface to include the `GetStudents` method to fetch the list of all the students in the university. Develop a service contract for the service using the following code:

```
[ServiceContract]
public interface IStudent
{
    [OperationContract]
    List<Student> GetStudents();
}
```

6. Add a new class named `Student` in your WCF Application. This class should hold student information, such as name, age, and stream. Make sure that this class is marked with the `DataContract` attribute and that all the public properties have the `DataMember` attribute. This is required so that when you return the list, the consumer of the WCF service can access the `Student` class and its members.

The following code snippet creates the `Student` class:

```
[DataContract]
public class Student
{
    [DataMember]
```

TAKE NOTE*

If you are returning the implicit datatypes, like int, string, and Boolean, you don't have to specify the DataContract and DataMember attributes.

```
public string Name
    { get; set; }
[DataMember]
public int Age
    { get; set; }
[DataMember]
public string Stream
    { get; set; }
}
```

7. Write the code for the GetStudents method in the StudentService.svc.cs file, which returns a list of all students enrolled in the university as shown:

```
public List<Student> GetStudents()
{
    List<Student> list = new List<Student>();
    list.Add(new Student() { Name = "Ruskin", Age=21,
    Stream="Science" });
    list.Add(new Student() { Name = "Abye", Age = 17, Stream =
    "Computer" });
    list.Add(new Student() { Name = "Sherry", Age = 22, Stream =
    "Management" });
    list.Add(new Student() { Name = "Mike", Age = 24, Stream =
    "Finance" });
    list.Add(new Student() { Name = "Julia", Age = 19, Stream =
    "Science" });
    return list;
}
```

8. Now that the service is implemented, the service name in the web.config file needs to be renamed to expose it. This is required because you have changed the name of the service contract and the .svc file. To expose the service, rename the service in the web.config file from Service to StudentService. Update the web.config file to reflect the change. Change the name attribute in the service node to SearchService. Change the contract attribute in the endpoint node to IStudent to match the interface name. The following code snippet shows how the ServiceModel section looks in the web.config file:

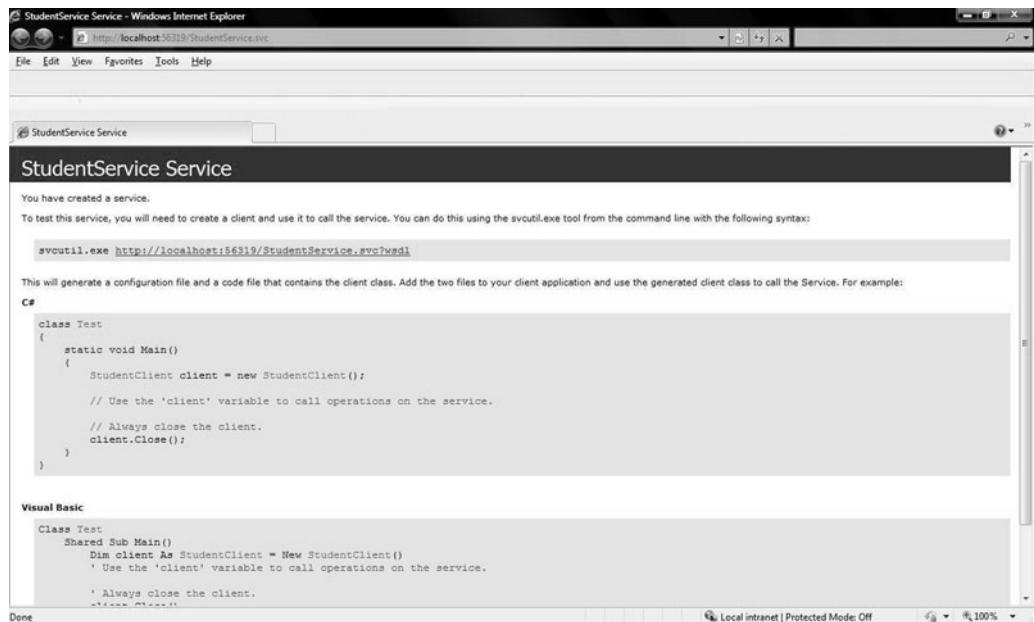
```
<system.serviceModel>
<services>
    <service behaviorConfiguration=
    "MyWCF.MyServiceBehavior" name="MyWCF.StudentService">
        <endpoint address="" binding="wsHttpBinding"
        contract="MyWCF.IStudent">
            <identity>
                <dns value="localhost" />
            </identity>
        </endpoint>
        <endpoint address="mex" binding="mexHttpBinding"
        contract="IMetadataExchange" />
    </service>
</services>
<behaviors>
    <serviceBehaviors>
        <behavior name="MyWCF.MyServiceBehavior">
            <serviceMetadata httpGetEnabled="true"/>
            <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
    </serviceBehaviors>
    </behaviors>
</system.serviceModel>
```

Now that you've created and configured the WCF service, you must verify whether the service is running correctly.

9. To verify the service, right click StudentService.svc and select the View In Browser option. This opens a new browser window that displays the service information as shown in Figure 10-5.

Figure 10-5

Service information



You have now created a WCF service that is hosted through ASP.NET. You can call this WCF service from any computer connected to the Internet.

Using a WCF Service

Once you have created a WCF service, you need to enable client applications to call and use the service.

WCF services can be called both synchronously as well as asynchronously.

The way you call the web service depends on the result expected from the web service. For example, a calculator web service would provide some result based on the operations you perform using the web service. If you do not receive the result, you will not be able to proceed with subsequent operations. Therefore, you would need to call the web service synchronously.

Again, consider the example of a web service that allows you to watch videos online or listen to music online. This needs to be called asynchronously. Once the call is complete, an event is raised that denotes whether the call was successful.

Let's see how you can first create a client application and then call the GetStudents WCF service from this application.



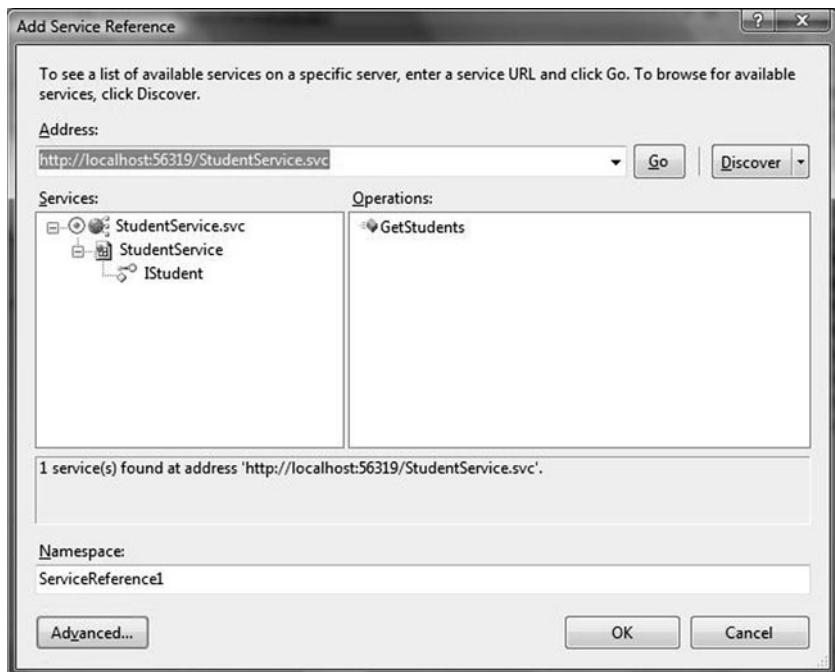
CALL A WCF SERVICE SYNCHRONOUSLY

To call a WCF service synchronously from a web application:

1. Start Visual Studio 2008.
2. Open the MyWCF solution.
3. Add a new web application project to it and name the project StudentServiceClient.
4. Create a reference to the StudentService WCF application by right clicking the StudentServiceClient Project tree node within Solution Explorer.
5. Select Add Service Reference to open the Add Service Reference dialog box as shown in Figure 10-6.

Figure 10-6

The Add Service Reference dialog box



6. Click Discover, select the Service.svc file from this project, and expand its associated tree node. The service contracts that are available through the service are displayed.
7. Expand the Services tree in the left pane to access the IStudent contract. Notice the namespace given by the dialog box—ServiceReference1. This is the default namespace given for the proxy class.
8. Click OK to add the service reference. Visual Studio will produce a new directory within the StudentServiceClient project directory named ServiceReferences. In this directory, Visual Studio generates information about the service in the form of XML files, XSD files, and a WSDL file (among others). You'll also get source code for a proxy class that will call the service on your behalf. By default, the proxy lands in a file named Reference.cs.
9. Now, you need to call the GetStudents service. First, create an instance of the ServiceReference1.StudentClient class to receive the results of calling the GetStudents method.
10. Add a DataGrid control on the web page as shown in this markup:

```
<form id="form1" runat="server">
<asp:DataGrid ID="dgStudentList" runat="server" />
</form>
```

11. Call the GetStudents method from the StudentClient class and then bind the result of the method to a DataGrid in the Page_Load method of the Default.aspx page as shown:

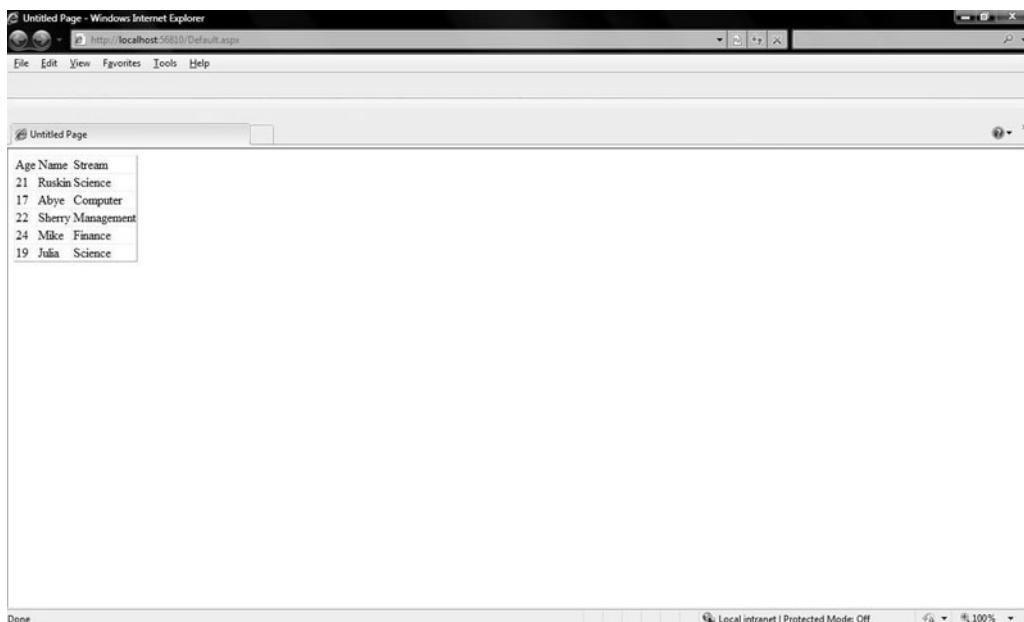
```
protected void Page_Load(object sender, EventArgs e)
{
    ServiceReference1.StudentClient client = new
    StudentServiceClient.ServiceReference1.StudentClient();
    dgStudentList.DataSource = client.GetStudents();
    dgStudentList.DataBind();
}
```

The previous code will display the list of students.

Now that you've created the client that calls the StudentService, you can run the program. Right click the file Default.aspx, and select View in the Browser option. The page displays a list of students in the data grid as shown in Figure 10-7.

Figure 10-7

Student list



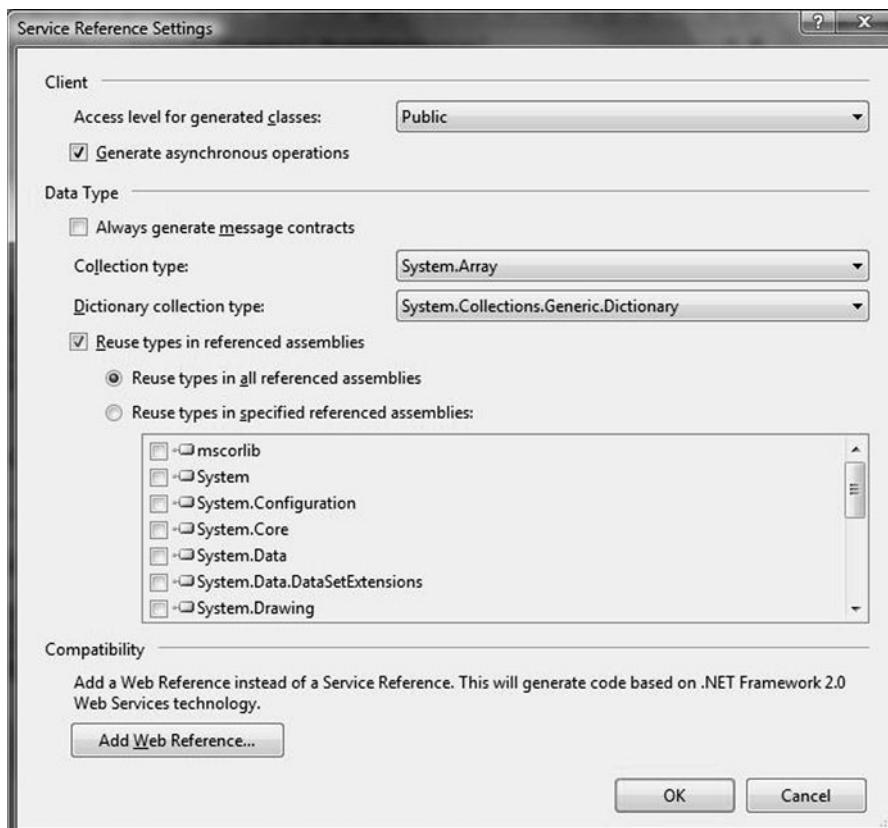
CALL A WCF SERVICE ASYNCHRONOUSLY

You can generate the reference for asynchronous calls as well. This is typically required when you are not expecting a result back from the service or when making the I/O operations that can block the thread. For example, you have a Student Registration service that will insert the student information in the database. In this situation, your service will be inserting the database and will not return any result. To call a WCF service asynchronously:

1. In the Add Service Reference dialog box, click the Advanced button. This opens the Service Reference Settings dialog box as shown in Figure 10-8.

Figure 10-8

The Service Reference Settings dialog box



2. Select the Generate Asynchronous Operations check box, and click OK. To call the service, you need to change the way you call the WCF GetStudents method.

3. Add the AsyncCompleted handler as shown in the following snippet:

```
client.GetStudentsCompleted += new EventHandler<StudentService
Client.ServiceReference1.GetStudentsCompletedEventArgs>
(client_GetStudentsCompleted);
```

You get the result of the method call in the `client_GetStudentsCompleted` method.

4. Now call the method asynchronously, as shown:

```
client.GetStudentsAsync();
```

5. Go to the AsyncCompleted method handler. You'll get the result in the `Result` property of the `GetStudentsCompletedEventArgsResult` object:

```
void client_GetStudentsCompleted(object sender,
ServiceReference1.GetStudentsCompletedEventArgs e)
{
    dgStudentList.DataSource = e.Result;
    dgStudentList.DataBind();
}
```

Hosting WCF Applications

You can host WCF applications on Windows Service or Internet Information Services (IIS). You can host WCF services with ASP.NET in either side-by-side mode or ASP.NET compatibility mode.

In the side-by-side mode, the WCF services hosted by IIS are located together with ASP.NET applications. The ASP.NET runtime handles only ASP.NET requests. WCF communicates using different protocols, including HTTP. The ASP.NET files and WCF services are stored in a common Application Domain (AppDomain). This enables ASP.NET to provide common infrastructure services, such as AppDomain management and dynamic compilation, for both WCF and the ASP.NET runtime. In this hosting mode, WCF and ASP.NET can share items such as static variables and public events.

In the ASP.NET compatibility mode, you can run your WCF application as a full-fledged ASP.NET program. All of WCF's functionality is available through ASP.NET. The design of WCF is such that it unifies the programming model across multiple transports and hosting environments. The WCF services running using ASP.NET compatibility mode have full access to the ASP.NET pipeline. WCF applications have access to the session state, `Server` object, `Response` object, and `Request` object of ASP.NET. To secure WCF applications in ASP.NET compatible mode, you can associate it to Windows access control lists (ACLs).

CERTIFICATION READY?

Call a Windows
Communication
Foundation (WCF) service
or a web service from an
ASP.NET web page.
3.3

■ Understanding ASP.NET Extensions



THE BOTTOM LINE

ASP.NET 3.5 introduces several new features to enable programmers to develop data-driven web applications with much less coding. In addition, ASP.NET 3.5 provides features that help developers create rich Internet-based applications.

Let's look at the new features that ASP.NET 3.5 provides:

- **ASP.NET Dynamic Data:** Enables you to build a fully customizable, data-driven application without writing code.
- **Model View Controller (MVC) Framework:** Allows you to create flexible, test-driven, loosely coupled applications with better separation of UI logic, model logic, and interaction logic.
- **Silverlight:** Helps create Rich Internet Application (RIA) over the web.

Along with these new features, ASP.NET also extends some of the existing features. The features that ASP.NET extends are:

- **The ASP.NET AJAX history support navigation feature:** Uses the Forward and Back buttons in the browser for navigation.
- **The ASP.NET AJAX script feature:** Reduces the number of scripts that are downloaded to the browser, which results in improved performance for the web application.
- **The ADO.NET Dynamic Data Services feature:** Provides new services that find, manipulate, and deliver data over the web using simple URLs. The benefits include an easy and flexible way to access data over the web, while enabling the separation of presentation and data access code.
- **The ADO.NET Entity Framework:** Works as a new modeling framework providing a conceptual model of a database schema that closely aligns to a real-world view of the information. This enables you to easily understand and maintain application code that is shielded from the underlying database schema changes.

In the following sections, you'll learn about the new features in ASP.NET 3.5.

Using Dynamic Data

The Dynamic Data feature of ASP.NET enables you to develop datacentric web applications that can use the LINQ-to-SQL or Entities Framework data model at runtime. The Dynamic Data feature also determines how to render a UI based on the data model.

Using the LINQ-to-SQL data model, you can create types for the schema of the SQL Server database, which allows you to create LINQ queries for tables, views, and functions.

Using the ADO.NET Entity Framework, you can achieve loose coupling between the data model and the database. You can optimize the performance and consistency of the database schemas. This framework exposes an application-oriented data model that is loosely coupled to the database.

ASP.NET Dynamic Data provides a scaffolding framework, which you can use to create database-driven applications with built-in validations. You can easily customize the scaffolding framework and integrate scaffolding elements with the existing application. The advantage of using Dynamic Data is that you do not need to write very much code. You can use this framework to perform operations, such as creating, updating, removing, displaying, and validating data. You can easily separate the business logic of an application independent of the presentation. The scaffolding framework allows you to run an application based on the database schema. It also allows you to develop applications very easily and in very less time. At the same time, you can take advantage of the built-in validations of data.

Dynamic Data uses page templates and field templates to display data. You can customize these templates per your requirement to display rendered data. The dynamic data classes are stored in the `System.ComponentModel.DataAnnotations` and `System.Web.DynamicData` namespaces. Dynamic Data provides enhancements to data controls by adding dynamic behavior.

Dynamic Data provides predefined `DynamicControl` control templates that help display data dynamically and build the UI. You can customize these templates and reuse them across the application. The controls that Dynamic Data enhances are `DetailosView`, `GridView`, `FormView`, and `ListView`.

MORE INFORMATION

For more information on Dynamic Data and the scaffolding framework, refer to the ASP.NET Dynamic Data Overview section on MSDN.

Using Model View Controller

The ASP.NET Model View Controller (MVC) is an architectural pattern developed by Microsoft. MVC allows you to separate the input logic, UI logic, and business logic of the application. This separation helps in development, maintenance, and testing of each component independently.

MVC has the following components:

- **Model:** Handles the business logic. Model objects are the application's data domain. These objects are stored in and retrieved from the database. For example, the `Student` object can be retrieved from the database. You can perform operations on this object and update the information back to the database.

- **View:** Handles the UI logic. Views are typically the user interfaces and based on the model data.
- **Controller:** Handles the input logic and manages user interactions.

The ASP.NET MVC framework enables you to develop flexible, lightweight web applications that help you manage code for web applications. The MVC framework can be accessed from the `System.Web.Mvc` assembly. MVC segregates an application into three parts: Model, View, and Controller. Each Model is associated with a number of views, while a controller provides data inputs to the Model. Consider the example of an online shopping cart application. When you click a product, you can view that product's information. Here, the user interacts with the UI. When you select the product and click the Add button to add it to your shopping cart, the Controller handles your input, updates your shopping cart, and notifies the Model about this action. Now, you can view the contents of the shopping cart. In this case, the View uses the Model to display the UI.

MVC provides the following advantages:

- It provides better separation and loose coupling. In MVC, the View only displays information, the Controller manages and responds to user inputs and interactions, and the Model represents the data model for the application.
- MVC improves the testability of the application. You can run the unit test cases without running the controllers.
- The MVC framework supports ASP.NET routing and URL mapping.

Introducing Silverlight

Currently, there is a growing demand for rich Internet-based applications (RIA). To create RIAs, it is not sufficient to use traditional web tools, techniques, tricks, or even AJAX. To develop such applications, you need a cross-browser and cross-platform solution that helps browsers render a rich and interactive interface, such as Adobe Flash. It is possible to develop such applications using Microsoft Silverlight.

WORKING WITH SILVERLIGHT

Silverlight allows you to develop Microsoft .NET-based RIAs for the web. Silverlight is a cross-platform and cross-browser plug-in. It exposes the programming framework and features that are a subset of the .NET Framework and the Windows Presentation Foundation (WPF). Microsoft currently offers the Silverlight environment free for development of RIAs.

Silverlight 2.0 presents several improvements over the previous version. These include support for RIA controls, multithreading, improved security, and a higher download size for the packets. Silverlight applications are made up of text-based files with markup and code. These files can be created using any text editor. Visual Studio simplifies the task of creating these markups and code significantly.

UNDERSTANDING SILVERLIGHT ARCHITECTURE

Silverlight processes chunks of information on the client computer; therefore, it decreases the load on the server. This improves the overall performance of the web application. Silverlight achieves this using a plug-in that is installed on the client's computer. This plug-in is required to run Silverlight applications. All of Silverlight's functionality, including audio and video codecs, is encapsulated in this plug-in.

On the server side, Silverlight does not have any special requirements for hosting, just that the server should be able to host HTML documents.

To execute a Silverlight application, the complete Silverlight application must be downloaded from the server. This application may comprise many types of files. In some instances, for the application to function correctly, you may need to reassign the MIME types for XAML and XAP files to Silverlight.

USING SILVERLIGHT CONTROLS

Silverlight and WPF are similar in using Extensible Application Markup Language (XAML) to design the UI. Silverlight controls provide extensive and customizable functionality that allow you to tweak the control according to your requirements. You can use the following in your application:

- **Styles or templates:** Modify the look and feel of controls.
- **Event handlers:** Specify the behavior of the controls.

You can also create or derive your own custom controls for an application.



CREATE SILVERLIGHT APPLICATIONS

You can create Silverlight applications in your Web site. To create a sample application:

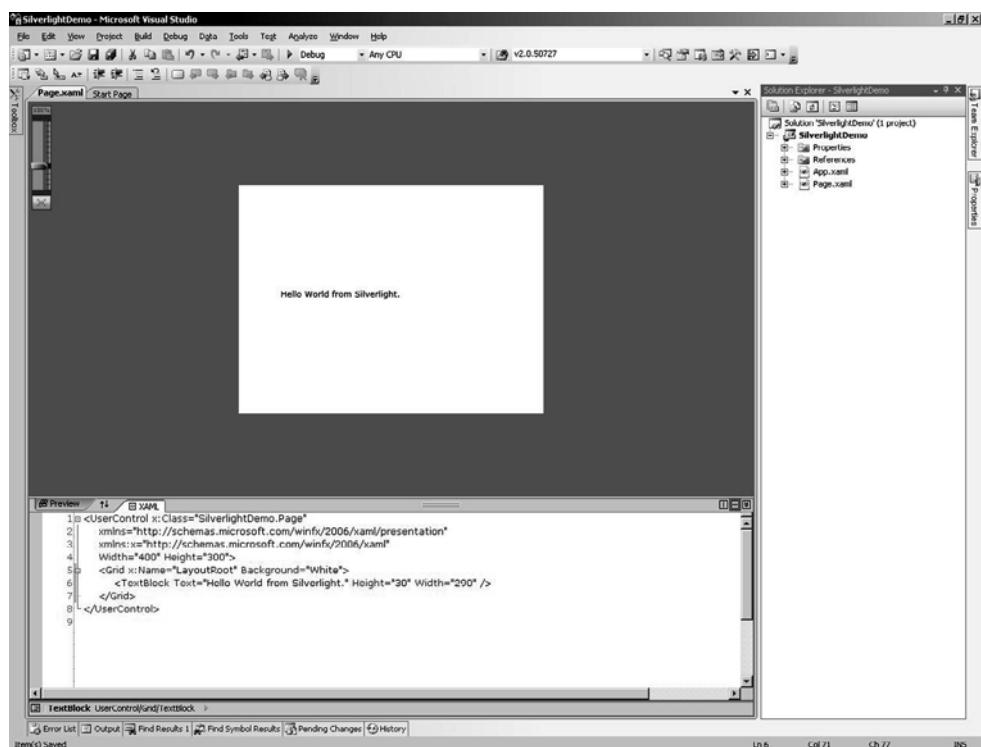
1. In Visual Studio 2008, click Create Project. The New Project window will be displayed.
2. Select C# project using the Silverlight Application Template.
3. Name the file SampleSilver.
4. Click OK. You will be prompted to generate either a Web Site or a web application or just a test page.
5. Select Test Page, and click OK. Visual Studio creates a sample application automatically for your test page named page.xaml. You can view the page.xaml by double clicking it in Solution Explorer. The markup of the Silverlight page will be as shown:

```
<UserControl x:Class="SilverlightDemo.Page"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="400" Height="300">
<Grid x:Name="LayoutRoot" Background="White">
<TextBlock Text="Hello World from Silverlight."
Height="30" Width="290" />
</Grid>
</UserControl>
```

Figure 10-9 displays the page in Visual Studio Design View.

Figure 10-9

The Silverlight Page in Design View





CREATE AND CONSUME AN XML WEB SERVICE

In this exercise, you will create a simple calculator service that will allow Add, Subtract, Multiply, and Divide operations on integers. Then, you'll call each of these methods from a web page and display the result.

To create the XML Web Service:

1. Create a new ASP.NET Web Service application, and name it SimpleCalculator. This will create Service1.asmx and Service1.asmx.cs files.
2. Rename these files to SimpleCalculator.asmx and SimpleCalculator.asmx.cs correspondingly.
3. Open the SimpleCalculator.asmx.cs file, and rename the class to SimpleCalculator. Make sure that you rename the value of the class name attribute in the .asmx file to SimpleCalculator. Your asmx file should look like this:

```
<%@ WebService Language="C#"
CodeBehind="SimpleCalculator.asmx.cs" Class="SimpleCalculator" %>
```

4. Rename the existing HelloWorld method to Add with two integer parameters, and change the return type to int. Add the code to return the addition in the Add method as shown here:

```
[WebMethod]
public int Add(int a, int b)
{return a + b;}
```

5. Similarly add methods for Subtract, Multiply, and Divide operations:

```
[WebMethod]
public int Subtract(int a, int b)
{return a - b;}
```

```
[WebMethod]
public int Multiply(int a, int b)
{return a * b; }
```

```
[WebMethod]
public int Divide(int a, int b)
{
    if(b != 0)
        return a / b;
    throw new DivideByZeroException();
}
```

6. Now create a new ASP.NET web application and add a new web form named Use-Calculator. In this form, add two text boxes to enter the operands and add four buttons for performing operations on these two operands. The markup for the aspx page is as follows:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<br />
<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
<br />
<br />
<asp:Button ID="Add" runat="server"
onclick="Add_Click" Text="Add" />
```

```
<asp:Button ID="Subtract" runat="server"
    onclick="Subtract_Click" Text="Subtract" />
<asp:Button ID="Multiply" runat="server"
    onclick="Multiply_Click" Text="Multiply" />
<asp:Button ID="Divide" runat="server"
    onclick="Divide_Click" Text="Divide" />
```

Note that you've added the `Click` event handlers for the buttons.

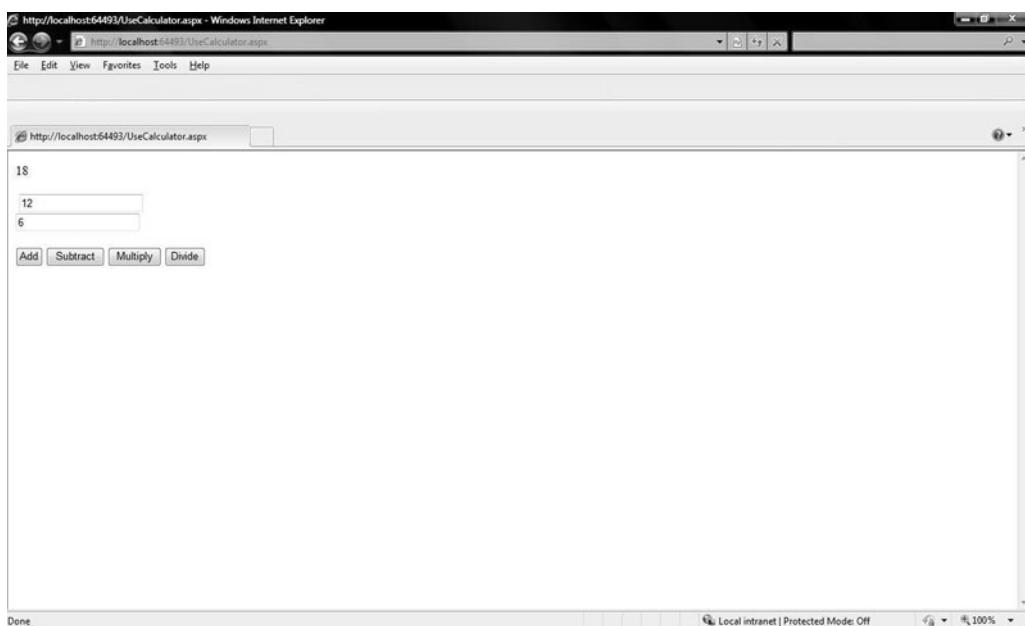
7. Add the web reference to the web service created earlier by selecting the Add Web Reference option and specifying the URL of your web service—`http://localhost/ SimpleCalculator/SimpleCalculator.asmx`.
8. Specify the namespace as `CalcProxy`.
9. Now open the code behind for the web page created in Step 6, add the code to call the web methods in the corresponding button `Click` event handler methods. The complete code for the code behind file is provided:

```
public partial class UseCalculator: System.Web.UI.Page
{
    CalcProxy.SimpleCalculator calc = new
    StudentASMXService.CalcProxy.SimpleCalculator();
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Add_Click(object sender, EventArgs e)
    {
        int result = calc.Add(int.Parse(TextBox1.Text),
        int.Parse(TextBox2.Text));
        Response.Write(result.ToString());
    }
    protected void Subtract_Click(object sender, EventArgs e)
    {
        int result = calc.Subtract(int.Parse(TextBox1.Text),
        int.Parse(TextBox2.Text));
        Response.Write(result.ToString());
    }
    protected void Multiply_Click(object sender, EventArgs e)
    {
        int result =
        calc.Multiply(int.Parse(TextBox1.Text), int.Parse(TextBox2.Text));
        Response.Write(result.ToString());
    }
    protected void Divide_Click(object sender, EventArgs e)
    {
        int result =
        calc.Divide(int.Parse(TextBox1.Text), int.Parse(TextBox2.Text));
        Response.Write(result.ToString());
    }
}
```

10. Run the web application. Enter the values 12 and 6 in the text boxes, and click the Add button. The result will be displayed on the form as shown in Figure 10-10.

Figure 10-10

The result



SKILL SUMMARY

Web services can be used across different domains, networks, and security barriers. In this lesson, you learned to create XML Web Services, and call these services by generating the proxy class. In addition, you learned how to call the WCF service synchronously and asynchronously.

This lesson discussed the extended features of ASP.NET, such as Dynamic Data, MVC framework, and Silverlight. Dynamic Data is used to create datacentric applications while MVC framework is used to create loosely coupled, robust, and lightweight applications. In this context, you learned about the three components of MVC: Model, View, and Controller. Finally, you learned that Silverlight is used to create rich Internet-based applications. You learned about the architecture of Silverlight also.

For the certification examination:

- Create and use web services in ASP.NET applications.
- Create and use XML web services.
- Use Windows Communication Foundation to build web applications based on service-oriented architecture.
- Use Silverlight, ASP.NET Model View Controller, and ASP.NET Dynamic Data.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. You can apply the _____ attribute to the web service without defining any parameters to identify the methods.
2. WCF services running using the _____ mode have full access to the ASP.NET pipeline
3. _____ unifies multiple Windows communication application programming interfaces (APIs).

4. The ASP.NET _____ allows you to create new data-driven Web sites with minimal coding.
5. _____ allows you to develop Microsoft .NET-based RIAs for the web.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---|--|
| T | F 1. The MVC framework supports ASP.NET routing and URL-mapping. |
| T | F 2. To secure WCF applications in ASP.NET compatible mode, you can associate it to Windows access control lists (ACLs). |
| T | F 3. The Model object manages user interactions and handles input logic. |
| T | F 4. The address element in WCF specifies how to transmit the message. |

Multiple Choice

Circle the letter that corresponds to the best answer.

1. Which of the following View controls uses predefined Dynamic Data templates to display data automatically?
 - a. CellView
 - b. TableView
 - c. DynamicView
 - d. GridView
2. Which of the following elements in WCF defines the way a message would be transmitted?
 - a. Behavior
 - b. Channels
 - c. Contract
 - d. Binding
3. You are administering authentication procedures in your organization. Which authentication method would make the security susceptible to monitoring tools?
 - a. Basic authentication
 - b. Windows digest
 - c. Client certificates
 - d. Encryption
4. Which of the following is the default message exchange pattern supported by WCF?
 - a. Duplex
 - b. Request-response
 - c. One-way
 - d. Custom

■ Case Scenario

Scenario 10-1: Using User and Custom Controls

Assume that you have created a WCF service that reads student information from the files on a server disk and returns the results to you. You are developing a web application that will consume this web service. How will you call this service?



Workplace Ready

Developing Web Services

You are creating a web application for a Library Management System. You want to develop a service that will return the list of all available books. The service will be called across the domains and across the network. How will you develop this service?

Working with Mobile Applications

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing mobile applications.	Access device capabilities.	6.1
Introducing mobile controls.	Control device-specific rendering.	6.2
Introducing mobile controls.	Add mobile web controls to a page.	6.3
Introducing mobile controls.	Implement control adapters.	6.4

KEY TERMS

control adapters

Microsoft Mobile Internet Toolkit (MMIT)

mobile controls

Wireless Markup Language (WML)

Mobile phones play a vital role in our day-to-day activities. New technologies have packed the services of phones, computers, cameras, and the Internet into one gadget—the mobile phone.

The convergence of personal computing and mobile telephony has led to the development of software applications for mobile phones. The .NET framework works with ASP.NET Mobile controls commonly known as ASP.NET for mobile, which is used to build mobile applications that support all types of mobile devices. ASP.NET mobile controls were earlier known as the *Microsoft Mobile Internet Toolkit (MMIT)*.

■ Introducing Mobile Applications



THE BOTTOM LINE

A mobile web application project is similar to a standard ASP.NET web application, except that it exhibits different form behavior to display content on a mobile device. Mobile web applications support many ASP.NET features, such as tracing and debugging. However, in mobile web applications, the web form's design view does not provide a strict graphical representation of the page. This is because each mobile phone exhibits different views and settings and the application must be flexible.

With .NET mobile, you can develop a single application that can run on different mobile devices having different browsers, resolution, and input methods.

The .NET Framework detects the browser being used on the mobile device using the `HttpContext.Current.Request.Browser` class. The `HttpBrowserCapabilities` object provided by this class has the property `IsMobileDevice`, which helps detect the browser. .NET mobile displays the content based on the browser detected. With .NET mobile, it is not necessary to build applications specific to each browser; so, ASP.NET reduces the

amount of work required for developers. The controls in .NET mobile render the appropriate markup language, such as HTML 3.2, WML 1.1, cHTML, or XHTML while dealing with different screen sizes, orientations, and mobile device capabilities.

Developing Mobile Applications

The demand for mobile applications is growing because today's mobile device is no longer used only for voice communication—it is a multifunctional device, as powerful as a laptop or desktop; in fact, some mobile devices are more powerful than many desktops and laptops. In this section, you will learn to develop applications that can be browsed using mobile devices.

A mobile application is made up of the following objects:

- **Mobile page:** Acts as the outermost shell and container for the mobile application. A mobile page is synonymous to a web page in a desktop browser.
- **Mobile form:** Contains one or more forms, and it can have mobile panels in it.
- **Mobile panel:** Acts as a container for other controls on a mobile page. You can use panels to group controls.
- **Mobile controls:** Similar to HTML controls on a web page, except that they run on the server. They are specially designed for mobile applications and act as building blocks for the development of mobile applications.

IMPLEMENTING A MOBILE APPLICATION WORKFLOW

A mobile application follows a workflow similar to that of a typical Windows application. A mobile application processes a client request in the following manner:

1. The client requests a web page, and the request is sent over the Internet.
2. IIS receives the request.
3. The .NET Framework handles the request.
4. ASP.NET compiles the requested page.
5. .NET mobile renders the request as WML, HTML, or compact HTML on the mobile device.
6. The requested web page is then displayed to the client.

In a standard web application, you need to first create a project, and then, add pages to the project. Each project consists of forms and controls. The code required to make the page function could be written at the server or the client side. Similarly, for a mobile application, you need to first create a mobile Web site project. You can base your application on a template, which can be selected from a predefined list. You can also design mobile UI pages by dragging controls, such as labels and text boxes, and dropping them on the form that represents the screen of the mobile device. The code representing the functionality of these applications can be written at the server or client side.

Here is a typical login screen example. A login screen has the ID and the password as inputs. Code for the markup looks like this:

```
<%@ Page Language="C#" AutoEventWireup="true"
Inherits="Sample.Login" Codebehind="Login.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.
UI.MobileControls" Assembly="System.Web.Mobile" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <body>
        <mobile:Form id="form1" Title="Login" runat="server">
            <mobile:Label ID="lblLogin" Runat="server"
Text="User Name"></mobile:Label>
            <mobile:TextBox ID="txtUserName"
Runat="server"></mobile:TextBox>
```

```

<mobile:RequiredFieldValidator id="RequiredFieldValidator1"
runat="server"
ErrorMessage="Please enter User Name."
ControlToValidate="txtUserName" />
<mobile:Label ID="lblPassword" Runat="server"
Text="Password"></mobile:Label>
<mobile:TextBox ID="txtPassword" Runat="server"
Password="True"></mobile:TextBox>
<mobile:RequiredFieldValidator id="RequiredFieldValidator2"
runat="server" ErrorMessage="Please enter Password."
ControlToValidate="txtPassword" />
<mobile:Command ID="cmdLogin" Runat="server"
OnClick="cmdLogin_Click" >Login</mobile:Command>
<mobile:Link ID="lnkforgotpwd" Runat="server" Text="Forgot
Password" NavigateUrl="Forgot Password.aspx"></mobile:Link>
</mobile:Form>
</body>
</html>

```

Code behind for the `Login.aspx.cs` page looks like this:

```

using System;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
namespace Sample
{
    public partial class Login: System.Web.UI.MobileControls.MobilePage
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void cmdLogin_Click(object sender, EventArgs e)
        {
        }
    }
}

```

In the previous example, the mobile web page has a form control as a container and contains the mobile controls, Label, TextBox, Command, RequiredFieldValidator, and Link.

In the `Login.aspx.cs` code-behind file, the `Login` class inherits from the `System.Web.UI.MobileControls.MobilePage` namespace rather than the regular `System.Web.UI.Page` namespace.

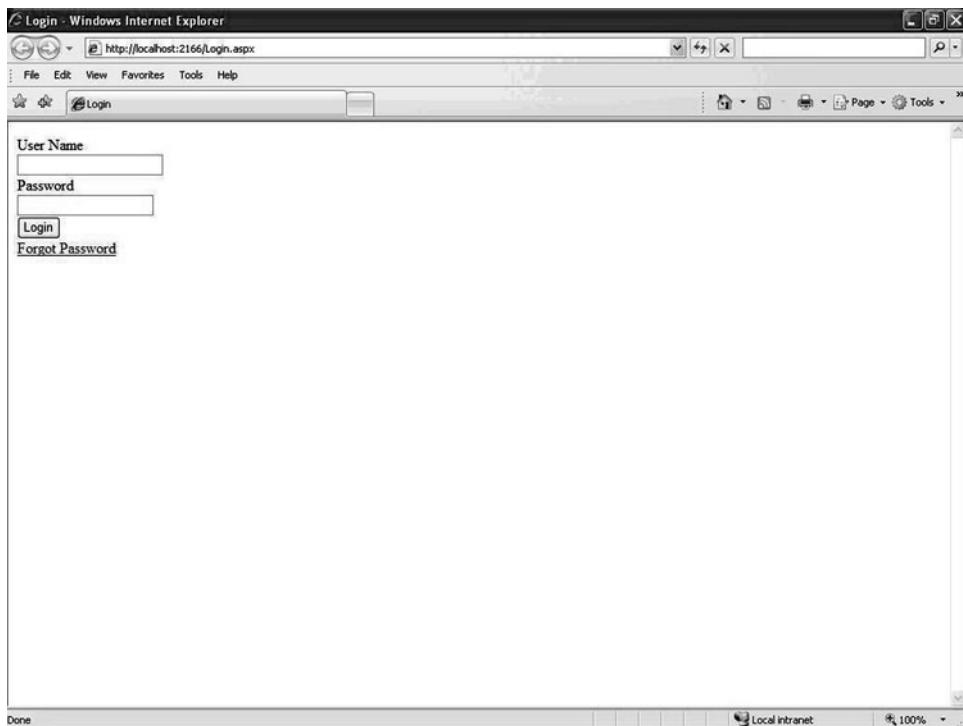
TAKE NOTE*

Note that `Inherits` is used within the `@Page` directive to specify the class that the current page should inherit from. In case the page inherits from the `MobilePage` object, then it should be specified as `Inherits="System.Web.UI.MobileControls.MobilePage"` namespace otherwise it will have a code-behind class, and you would need to mention the name of the class as shown in the earlier code (`Inherits="Sample.Login"`). Note that most of the code samples use pages that are inherited from the `MobilePage` class unless specified explicitly.

After you create a mobile page, you can view it using a browser on a supported device. Figure 11-1 shows the output.

Figure 11-1

A mobile page



CREATING A MOBILE WEB APPLICATION

A mobile web application is an ASP.NET Web site that contains mobile web pages or forms. You can use Microsoft Visual Studio to create ASP.NET mobile web pages.

Visual Studio provides tools for creating mobile web pages and their code and for managing the application containing your mobile web pages.

TAKE NOTE*

There are two main types of mobile applications: web based and local. The first type runs on the server, typically the web server, and is accessed by mobile devices through the Internet. This is where MMIT comes in. The second type is a standalone application running on the device itself, with or without Internet access. These devices are also known as smart devices. For this type of application, Microsoft provides a scaled-down version of the .NET Framework—the .NET Compact Framework (.NET CF). This lesson specifies how to work with mobile web applications; so, you will create ASP.NET Web sites in these examples.



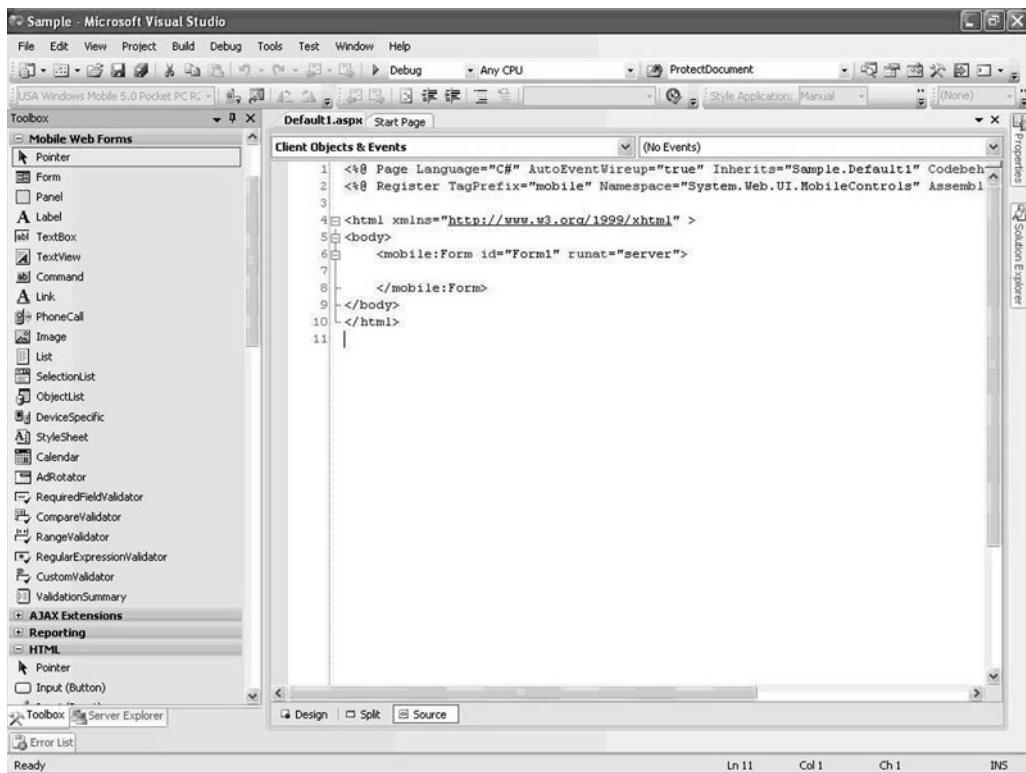
CREATE A MOBILE WEB APPLICATION

To create a simple mobile web application:

1. Create an ASP.NET Web site using C#. To do this, select Open Visual Studio 2008 > Create New Project > File > New > Project.
2. Under Visual C#, select Web.
3. From the Templates panel, select ASP.NET Web Application template.
4. Click OK.
5. In the code behind, include the `System.Mobile.UI` namespace.
6. Delete the `Default.aspx` page that is added by default to the Web site template.
7. Add the required mobile form to the project. To do this, go to the Solution Explorer, right click on the Web site, click on Add New item, and select Mobile Web Form.
8. Add the relevant code to the code-behind pages as shown in Figure 11-2.

Figure 11-2

Creating a mobile web application

**TAKE NOTE***

Once you create the mobile Web form, you can add mobile controls to the form. You will learn about adding mobile controls in the next section.

If you use server-side coding, the mobile controls run on the server and send the markup language to the browser. This markup specifies how to display the controls and content in the current form or page.

If you use client-side coding and the client scripts refer to specific Web server controls, they must use the identifier passed in the markup language because the identifiers vary based on the markup language. To retrieve the exact name of the control, you need to first compile the application, browse to the page or form that contains the control, and then view the source markup.

Understanding Device-Specific Rendering

When a web application is developed, it is typically designed for specific web browsers. Developers write the code keeping one browser or a set of specific web browsers in mind. If a specific web browser is not supported, then the web application may not behave the way it would in the supported web browsers. In a similar way, web pages or applications that are designed to run on desktops may not display the information correctly. You need to specifically add the ASP.NET mobile controls to render web pages automatically for mobile devices. The mobile controls have the ability to render the information differently for different sets of mobile devices, which can range from PDAs to web-enabled mobile phones.

There are two different methods that can be used for device-specific rendering:

- **Browser-specific properties:** You can use the `Request.Browser` property to query browser-specific properties. The information will be displayed based on the browser property that is generated.
- **Device-specific controls:** You can use the `DeviceSpecific` control to render the information for a specific mobile device. You must embed the device-specific control in a mobile form or in a mobile control.

The `web.config` file for your Web site will contain the filtering information. Based on the filter that is applied, the relevant information will be formatted for the mobile device.

USING DEVICE FILTERS

Device filters are used to customize the display of a web application based on the mobile device capability. You can store device filters in the `<deviceFilters>` section of the `web.config` file. There are two types of device filters that you can use:

- **Comparison-based filter:** Checks for a specific capability of a mobile device. You define a value for the filter that is compared with the value assigned to the device's capability. The comparison is performed at the runtime.
- **Evaluator-delegate-based filter:** Performs complex comparisons between the developer-specified class and the device capabilities. This type of filter is also known as an evaluator filter, and it works by using the evaluation method from a class, which is specified by the developer.

USING THE FILTER PROPERTY

The `Filter` property is used for evaluation of device filters that are listed in the `web.config` file for your web application. You need to define the `Filter` property in the `<deviceFilters>` section. The `Filter` property can be used to evaluate the device filters against the device capabilities of the `MobileCapabilities` class. The `Filter` name can be set to any of the following:

- A method defined on the page
- A device filter defined in the `web.config` file
- An associated `.ascx` file

The `web.config` file in the `system.web` section shows you how to set the comparison-based and the evaluator-delegate-based `Filter` property in the `web.config` file.

For an evaluator-delegate-based filter, set the `type` attribute to a class name and the `method` attribute to the method name of the specified class.

For a comparison-based filter, the required attributes are the `compare` property, which is set to the property that the evaluator needs to evaluate and the `argument` attribute against which the property `Namevalue` is compared. The following code shows the use of the `compare` and `argument` properties. In the absence of an argument name, a null value is used for comparison:

```
<system.web>
    <deviceFilters>
        <filter name="capability" type="SomeClassName"
method="SomeMethodName" />
        <filter name="capability" compare="ComparecapabilityName"
argument="Compareargument" />
    </deviceFilters>
</system.web>
```

USING THE `<DEVICE-SPECIFIC><CHOICE>` CONSTRUCT

The `<Device-Specific><Choice>` construct consists of two different elements: the `<Device-Specific>` element, which is mainly used for Mobile Controls Device Specific Rendering, and the `<Choice>` element. A control can contain only one `<DeviceSpecific>` element.

USING THE CHOICE ELEMENT

The `<DeviceSpecific>` element is used for placing the `<Choice>` elements. You can define multiple `<Choice>` elements within a single `<DeviceSpecific>` element. Each `<Choice>` element carries a set of different capabilities, which are used for evaluation against the device capabilities. This is how the `<Choice>` element works. To add device-specific markup for a control, you add the `<DeviceSpecific>` element as a child of the control. Within a `<DeviceSpecific>` element, you typically specify one or more `<Choice>` elements each containing attributes that specify how to evaluate the choice against target device capabilities. Multiple `<Choice>` elements can be part of a single `<DeviceSpecific>` element.

Each `<Choice>` element is applied in the order it is listed in the `<DeviceSpecific>` element. The `<Choice>` element is evaluated by the filter name against the targeted mobile device. When the filter matches, the evaluation process is terminated and the `<Choice>` element is used.

The following HTML code for a mobile Image control shows you how to set the `<Choice>` element:

```
<mobile:Image runat=server ImageURL="bw.gif">
<DeviceSpecific>
    <Choice Filter="isColor" ImageURL="HomeImg.gif"
        AlternateText="This device cannot display the image." />
    <Choice Filter="isWML11" ImageURL="DetailsImage.wbmp" />
    <Choice ImageURL="MoreDetailsImage.gif" />
</DeviceSpecific>
</mobile:Image>
```

CERTIFICATION READY?
Control device-specific rendering.
6.2

In this code, three `<Choice>` elements for the three elements are displayed. You can choose from these images. Based on the device, the selected image supported by that device will be displayed.

Viewing and Testing Mobile Web Applications

After you create an application, it is crucial that you test it to ensure that the application is performing as expected, the output is as desired, and there are no inconsistencies in the application behavior. You can build the same expectations when developing web applications by using ASP.NET.

You should expect that mobile pages must adhere to the standards just like any regular application; therefore, it is crucial for you to test it against the mobile devices for which it is designed. There are different methods that can be used to test mobile web pages. The method that you use largely depends on your requirements.

The two most commonly used methods are web browser on a desktop system and an emulator. Besides these two, you can also use wireless devices and Pocket PCs to test your mobile web pages.

USING A WEB BROWSER

Using a web browser on a desktop system is one way to view mobile web applications. Visual Studio 2008 has default support for Internet Explorer, but you can also configure Visual Studio to use any other web browser that is needed. For instance, you may be using Mozilla Firefox as the default web browser. If you right click on the web page and select Browse, you will get a window that lists the installed browsers. You can check the check box at the bottom of the window to set the default browser.

You can also use emulators to view your mobile web applications. An emulator is software that simulates the functionality of the physical mobile devices. Using web browsers has a number of benefits over emulators—you can easily trace the logical errors and validate the formatting, which may not be truly applicable if the application is designed for a mobile device.

USING AN EMULATOR

Consider a situation where you need to test an application for different mobile devices from four different manufacturers. It would be difficult for you to collect so many different models of mobile devices. This is where you can use an emulator. Emulators are largely used to test applications that are designed for mobile devices. When using an emulator, you don't need the actual mobile device. Complete testing can be simulated on the emulator, and results are as accurate as they would have been on the real mobile device. The Microsoft Device Emulator (Figure 11-3) can be downloaded from <http://www.microsoft.com/downloads>.

Figure 11-3

Microsoft Device Emulator

**CERTIFICATION READY?**

Access device capabilities.

6.1

Different devices have different screen sizes or resolution. You need to adjust the web page so that it fits on the device screen. To do so, you need to enable the Fit to screen feature on the Emulator device browser for best viewing. This is a device property, and it may vary from device to device.

Understanding Mobile Web Forms

A mobile web form represents a number of mobile controls. When you build a mobile page, you need to ensure that you have a minimum of one mobile form. However, there is no limitation on the number of mobile forms that you can have. You can also have more than one mobile form on a mobile page. The display area on a mobile device is small. Therefore, loading lots of information on a single form may make it unreadable or make the page bulky. Use multiple forms to simplify and make it more user friendly.

The difference between a mobile form and a regular ASP.NET form is that it has a mobile web server control that defines the behavior of the form to be accessible through the mobile devices. A mobile web server control is added from the `System.Web.Mobile` namespace. Each mobile web server control has a number of adapters that are specifically designed for mobile devices.

The inheritance hierarchy of a Form mobile control per object-oriented notation is:

```
System...:::Object  
  System.Web.UI...:::Control  
    System.Web.UI.MobileControls...:::MobileControl  
      System.Web.UI.MobileControls...:::Panel  
        System.Web.UI.MobileControls...:::MobilePage
```

The following code will help you understand how to navigate from Form1 to Form2 and use multiple forms within a single page. Code A shows the code for a mobile page that has one Mobile form, Form1:

```
<%@ Page Language="C#" AutoEventWireup="true"  
Inherits="Sample.TwoForms" Codebehind="TwoForms.aspx.cs" %>  
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"  
Assembly="System.Web.Mobile" %>  
<html xmlns="http://www.w3.org/1999/xhtml" >  
  <body>  
    <mobile:Form id="Form1" runat="server">  
      <mobile:Label ID="Label1" Runat="server">This is  
      Form 1</mobile:Label>  
      <mobile:Link ID="Link3" Alignment="Right" Runat="server"  
      Text="Home" NavigateUrl="Home.aspx"></mobile:Link>
```

```

<mobile:Link ID="lnkforgotpwd" Runat="server"
Text="Form2" NavigateUrl="#Form2"></mobile:Link>
</mobile:Form>
</body>
</html>

```

Code B shows the code for a mobile page that has two Mobile forms, Form1 and Form2:

```

<%@ Page Language="C#" AutoEventWireup="true"
Inherits="Sample.TwoForms" Codebehind="TwoForms.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
<mobile:Form id="Form1" runat="server">
<mobile:Label ID="Label1" Runat="server">This is
Form 1</mobile:Label>
<mobile:Link ID="lnk" Runat="server" Text="Form2"
NavigateUrl="#Form2"></mobile:Link>
<mobile:TextBox ID="txtName"
Runat="server">Name</mobile:TextBox>
</mobile:Form>
<mobile:Form id="Form2" runat="server"
OnActivate="Form2_Activate">
<mobile:Label ID="Label2" Runat="server">This is
Form 2</mobile:Label>
<mobile:Link ID="Link1" Runat="server" Text="Back"
NavigateUrl="#Form1"></mobile:Link>
<mobile:Label ID="Label3"
Runat="server">Message</mobile:Label>
</mobile:Form>
</body>
</html>

```

This is the code behind for the `TwoForms.aspx.cs` page:

```

using System;
using System.Web.Mobile;
using System.Web.UI.MobileControls;
namespace Sample
{
    public partial class TwoForms: System.Web.UI.MobileControls.
MobilePage
    {
        string name;
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void cmdForm_Click(object sender, EventArgs e)
        {
            name = txtName.Text;
            ActiveForm = Form2;
        }
        protected void Form2_Activate(object sender, EventArgs e)
        {
            Label3.Text = "Welcome " + name;
        }
    }
}

```

The code in Code B is an extension of the Code A example. Here, the mobile web page has two mobile web forms, Form1 and Form2. The default form is Form1, and a link to Form2 is provided in Form1. When this link is clicked, Form2 will be activated and displayed in the same mobile web page. The forms are not displayed simultaneously.

Note that a mobile page is like a deck of cards with each card being a form; but only one card can be on top at a time. Therefore, you can display one form at a time and not all the forms simultaneously.

■ Introducing Mobile Controls



Using ASP.NET mobile controls, you can generate **Wireless Markup Language (WML)**, compact HTML (cHTML), HTML, and XHTML content for different mobile devices. These controls allow you to target a wide range of devices, including web-enabled mobile phones, pagers, and personal digital assistants (PDAs) such as Pocket PCs.

TAKE NOTE*

Desktop ASP.NET applications use the machine.config file to maintain device and browser information.

ASP.NET mobile controls consist of server controls and device adapters that can intelligently render pages and controls on a mobile device. They also extend the schema of the machine.config file and add elements to support page rendering in mobile devices. ASP.NET also provides extensibility in mobile programming so that users can add new customized controls and support for new devices.

Understanding Types of Mobile Controls

Most of the controls used in a mobile web page are similar to ASP.NET web server controls. For example, the behavior of the `System.Web.UI.MobileControls.Label` and `System.Web.UI.MobileControls.TextBox` controls is similar to that of the ASP.NET `System.Web.UI.WebControls.Label` and `System.Web.UI.WebControls.TextBox` controls.

Like standard web controls, mobile controls are also categorized into groups based on their functionality, for example:

- Mobile utility controls
- Input validation controls
- UI controls

Table 11-1 lists some common mobile utility controls and their functions.

Table 11-1

Mobile utility controls

CONTROL	FUNCTION
AdRotator	Displays an advertisement.
Calendar	Displays a calendar.
PhoneCall	Provides the ability to call a mobile number.

Table 11-2 lists some common input validation controls and their functions.

Table 11-2

Input validation controls

CONTROL	FUNCTION
CompareValidator	Compares the values between two input controls or an input control with a fixed value.
CustomValidator	Allows you to write a custom method to validate the entered value.

(continued)

Table 11-2 (continued)

CONTROL	FUNCTION
RangeValidator	Checks whether the input value falls within a specified range.
RegularExpressionValidator	Validates the input value against a specified pattern.
RequiredFieldValidator	Checks whether a mandatory field is not NULL.
ValidationSummary	Provides a summary of all the validation errors that occurred in a page.

Table 11-3 lists some common UI mobile controls and their functions.

Table 11-3

Common UI mobile controls

CONTROL	FUNCTION
Command	Posts the input values entered by the user from the controls on the page to the server.
Form	Provides a container for grouping mobile controls.
FontInfo	Provides font-related information.
Image	Displays an image.
Label	Displays text.
Link	Provides a link to another mobile form or URL.
List	Displays a list of items.
MobilePage	Defines a base class for all mobile pages.
ObjectList	Defines a list of objects.
Panel	Provides a container for other controls.
SelectionList	Defines a list of items from which the user can select a value.
StyleSheet	Defines styles that can be applied to other controls in a mobile page.
TextBox	Provides a box that accepts one line of input.
TextView	Provides a box that accepts multiple lines of input.

MORE INFORMATION

For more information about mobile controls and elements, refer to the `System.Web.UI.MobileControls` namespace section on MSDN.

In .NET Mobile, to define the details of mobile controls, you can use certain special elements such as `<Choice>`, `<Command>`, `<DeviceSpecific>`, `<Field>`, `<Item>`, `<PagerStyle>`, and `<Style>`.

Working with Mobile Controls

In this section, we look at some examples of how to use the commonly used mobile controls in mobile applications.

Here's an example that uses .NET Mobile to create a mobile application with mobile controls. This application is a simple login screen. In this example, the login screen creates a mobile web forms page with a single form on it. The form contains a Label, a TextBox, and a Command control. The markup for the example is shown next:

```
<%@ Page Language="C#" AutoEventWireup="true"
Inherits="Sample.Login" Codebehind="Login.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<body>
<mobile:Form id="form1" Title="Login" runat="server">
    <mobile:Label ID="lblLogin" Runat="server"
    Text="User Name"></mobile:Label>
    <mobile:TextBox ID="txtUserName"
    Runat="server"></mobile:TextBox>
    <mobile:RequiredFieldValidator id="RequiredFieldValidator1"
    runat="server" ErrorMessage="Please enter User Name."
    ControlToValidate="txtUserName" />
    <mobile:Label ID="lblPassword" Runat="server"
    Text="Password"></mobile:Label>
    <mobile:TextBox ID="txtPassword" Runat="server"
    Password="True"></mobile:TextBox>
    <mobile:RequiredFieldValidator id="RequiredFieldValidator2"
    runat="server" ErrorMessage="Please enter Password."
    ControlToValidate="txtPassword" />
    <mobile:Command ID="cmdLogin" Runat="server"
    OnClick="cmdLogin_Click" >Login</mobile:Command>
    <mobile:Link ID="lnkforgotpwd" Runat="server"
    Text="Forgot Password" NavigateUrl="Forgot
    Password.aspx"></mobile:Link>
</mobile:Form>
</body>
</html>
```

The code-behind file contains the following code:

```
using System;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;

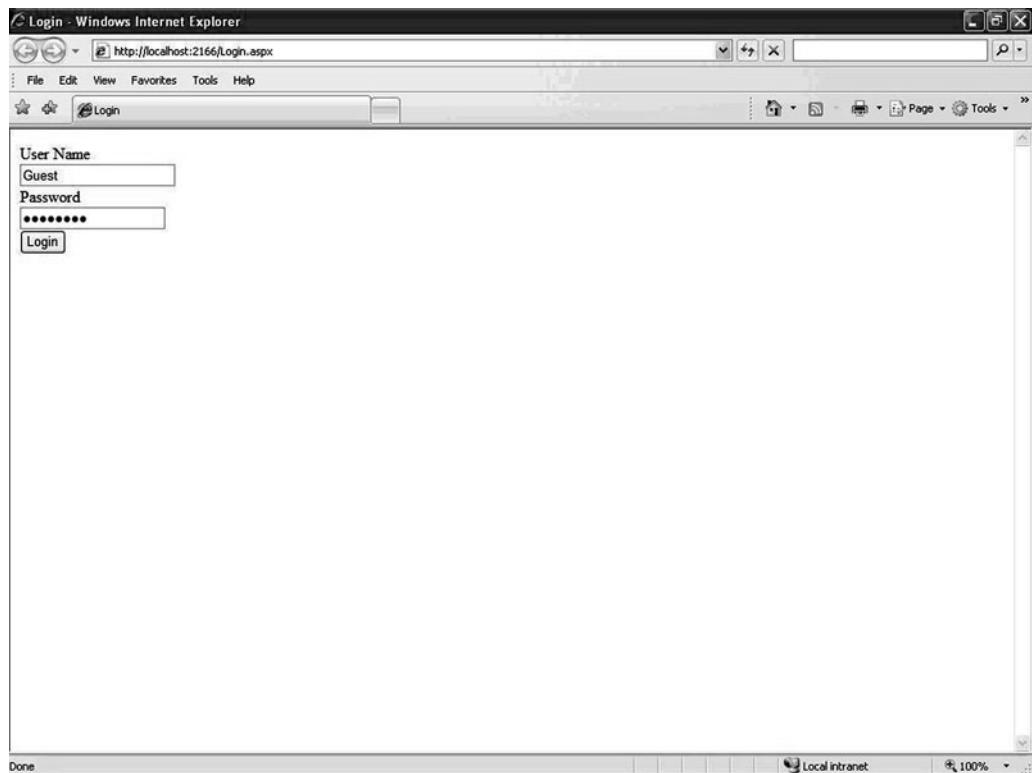
namespace Sample
{
    public partial class Login:
        System.Web.UI.MobileControls.MobilePage
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void cmdLogin_Click(object sender, EventArgs e)
        {
            if (!String.IsNullOrEmpty(txtPassword.Text))
            {
                if (txtPassword.Text == "password")
                {
                    Response.Redirect("Home.aspx?un=" + txtUserName.Text);
                }
            }
        }
    }
}
```

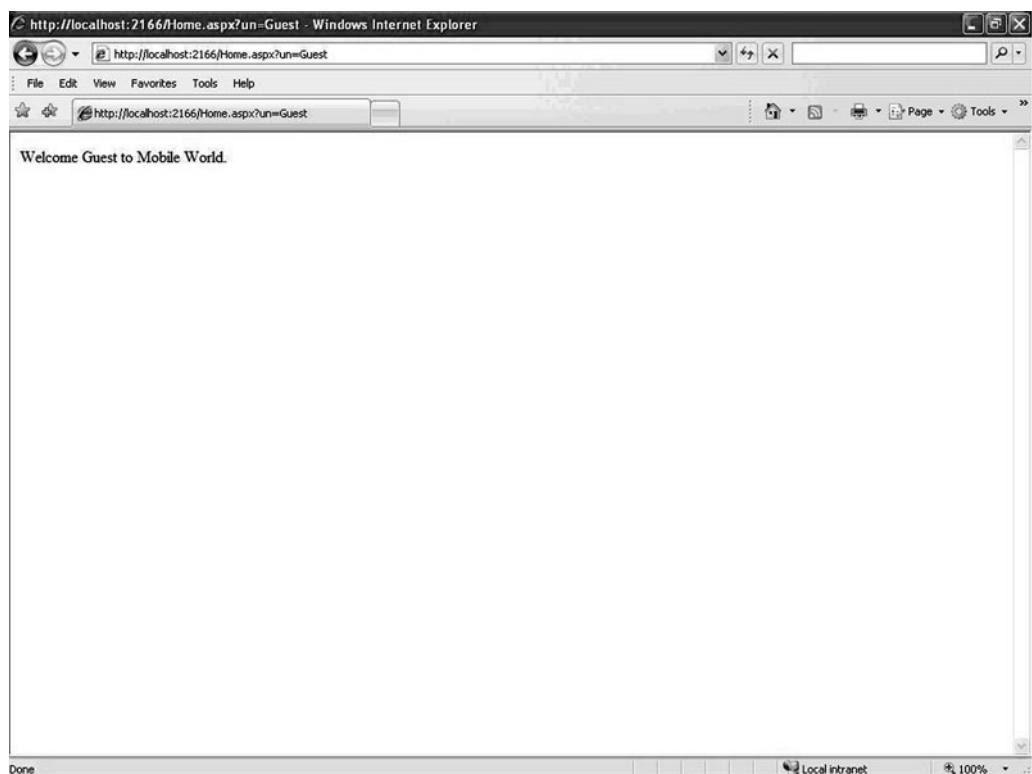
Note that all mobile controls must have the `runat` attribute set to `server` in order to secure proper rendering of the page for different devices. Figure 11-4 and Figure 11-5 shows the output of the login screen in a web browser.

Figure 11-4

The login screen

**Figure 11-5**

Login confirmation



In Figure 11-5, you can see how the code renders in a web browser.

To render correct output on a requesting mobile device, mobile controls need some information about the device, such as markup language supported, browser, cookie support, and number of lines that can be displayed.

The .NET Mobile Toolkit adds these mobile device capabilities to the server's machine.config file. Let's discuss some common controls that you can use in your mobile application.

USING THE CONTAINER CONTROL

With the help of a container control, you can group the contents and the controls to make logical sense in an ASP.NET mobile web forms page. Grouping the content and controls applies a specific style to the group. Mobile container controls are of two types: Form or Panel.

Mobile web form controls are a specialized form of ASP.NET web forms. A mobile web page must have at least one Form control <mobile:Form> tag. A Form control can also have many mobile Web controls while a mobile web page can have multiple Form controls unlike ASP.NET web pages.

The Panel control for mobile web pages provides device-specific rendering with the help of the ContentTemplate device templates and replaces the panel depending on the device.

The following code sample shows how to use the <mobile:Form> and <mobile:Panel> container controls:

```
<mobile:Form id="Form1" runat="server">
    <mobile:Label ID="Label1" Runat="server">Label</mobile:Label>
    <mobile:Panel ID="Panel1" Runat="server">
        <DeviceSpecific>
            <Choice>
                <ContentTemplate>
                    ...
                </ContentTemplate>
            </Choice>
        </DeviceSpecific>
    </mobile:Panel>
</mobile:Form>
```

USING THE COMMAND CONTROL

To understand the use of a Command control, study the next code example, which shows how to create a mobile page with two forms. By default, the first form is active. The UI of this form contains a label, an input box that accepts a name input, and a Submit button. The Click event in the first form displays the second form, which displays a welcome message. The Command control is used on the Submit button, which displays a page when clicked. The following code creates a page with two forms:

```
<%@ Page Language="C#" AutoEventWireup="true"
Inherits="Sample.TwoForms" Codebehind="TwoForms.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.
UI.MobileControls" Assembly="System.Web.Mobile" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <body>
        <mobile:Form id="Form1" runat="server">
            <mobile:Label ID="Label1" Runat="server">This is
Form 1</mobile:Label>
            <mobile:Label ID="lblName" Runat="server">Please enter
Name</mobile:Label>
            <mobile:TextBox ID="txtName"
Runat="server"></mobile:TextBox>
            <mobile:Command ID="cmdForm" Runat="server"
OnClick="cmdForm_Click">Welcome</mobile:Command>
```

```

<mobile:Link ID="lnkforgotpwd" Runat="server" Text="Form2"
NavigateUrl="#Form2"></mobile:Link>
</mobile:Form>
<mobile:Form id="Form2" runat="server"
OnActivate="Form2_Activate">
<mobile:Label ID="Label2" Runat="server">This is
Form 2</mobile:Label>
<mobile:Label ID="Label3" Runat="server"></mobile:Label>
<mobile:Link ID="Link1" Runat="server"
Text="Back" NavigateUrl="#Form1"></mobile:Link>
</mobile:Form>
</body>
</html>

```

The code-behind file looks like this:

```

string name;
protected void cmdForm_Click(object sender, EventArgs e)
{
    name = txtName.Text;
    this.ActiveForm = Form2;
}
protected void Form2_Activate(object sender, EventArgs e)
{
    Label3.Text = "Welcome " + name;
}

```

Additional code for code-behind file is shown next:

```

public partial class TwoForms : System.Web.UI.MobileControls.MobilePage
{
    string name;
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void cmdForm_Click(object sender, EventArgs e)
    {
        name = txtName.Text;
        this.ActiveForm = Form2;
    }
    protected void Form2_Activate(object sender, EventArgs e)
    {
        Label3.Text = "Welcome " + name;
    }
}

```

The code creates two forms where the first form accepts the name of the visitor. On clicking the button on Form 1, it navigates to Form 2 where a welcome message is displayed for the visitor.

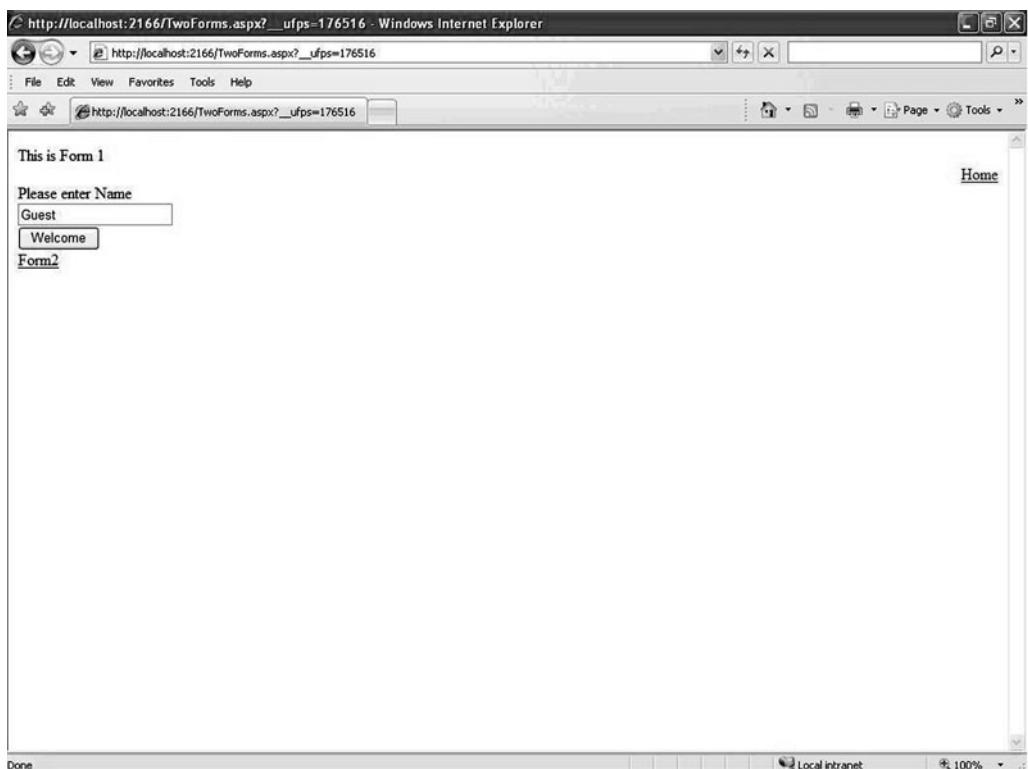
By default, the first form is opened. The first form has a label with the text `Please enter Name:`, an input box to input the name, and a Welcome button.

The second page is activated by the Submit button click on the first page and displays a greeting.

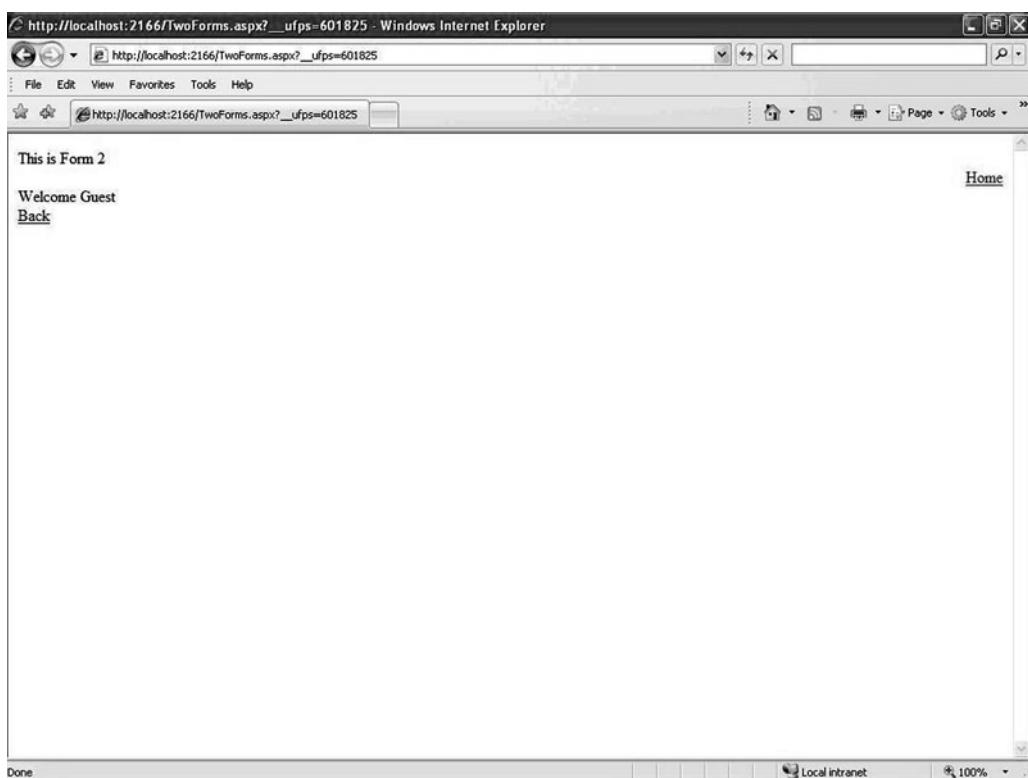
When the application runs on a mobile device, the two pages will display the output as shown in Figure 11-6 and Figure 11-7.

Figure 11-6

Using the command control—
Output displaying Form 1

**Figure 11-7**

Using the command control—
Output displaying Form 2



USING THE LIST CONTROL

Suppose you want to create a form that lists tasks on the first page. On the second page, you want to display the status of those tasks. This page is activated when a task is selected on the first page. The following code will help you achieve the desired result:

```
<%@ Page Language="C#" AutoEventWireup="true" Inherits="Sample.Lists"  
Codebehind="Lists.aspx.cs" %>
```

```

<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
    <body>
        <mobile:Form id="Form1" runat="server">
            <mobile:Label ID="lblHeader" Runat="server">List Control
Demo</mobile:Label>
            <mobile:Label ID="lblContentTasks" Runat="server">List
of Tasks</mobile:Label>
            <mobile:List ID="lstTasks" OnItemCommand ="Show_TaskStatus"
Runat="server">
                </mobile:List>
            </mobile:Form>
            <mobile:Form ID="Form2" Runat="server">
                <mobile:Label ID="lblContent" Runat="server"></mobile:Label>
            </mobile:Form>
        </body>
    </html>

```

You need to add the following code to the code-behind.

```

using System;
using System.Collections;
using System.Web.Mobile;
using System.Web.UI.MobileControls;

namespace Sample
{
    public partial class Lists:
        System.Web.UI.MobileControls.MobilePage
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                lstTasks.DataSource = GetTasks();
                lstTasks.DataBind();
            }
        }

        private ArrayList GetTasks()
        {
            ArrayList Tasks = new ArrayList(5);
            Tasks.Add((object)"Understand Windows Mobile Development");
            Tasks.Add((object)"Mobile Web Development");
            Tasks.Add((object)"Implement Sample");
            Tasks.Add((object)"Run Application in Simulator");
            return Tasks;
        }

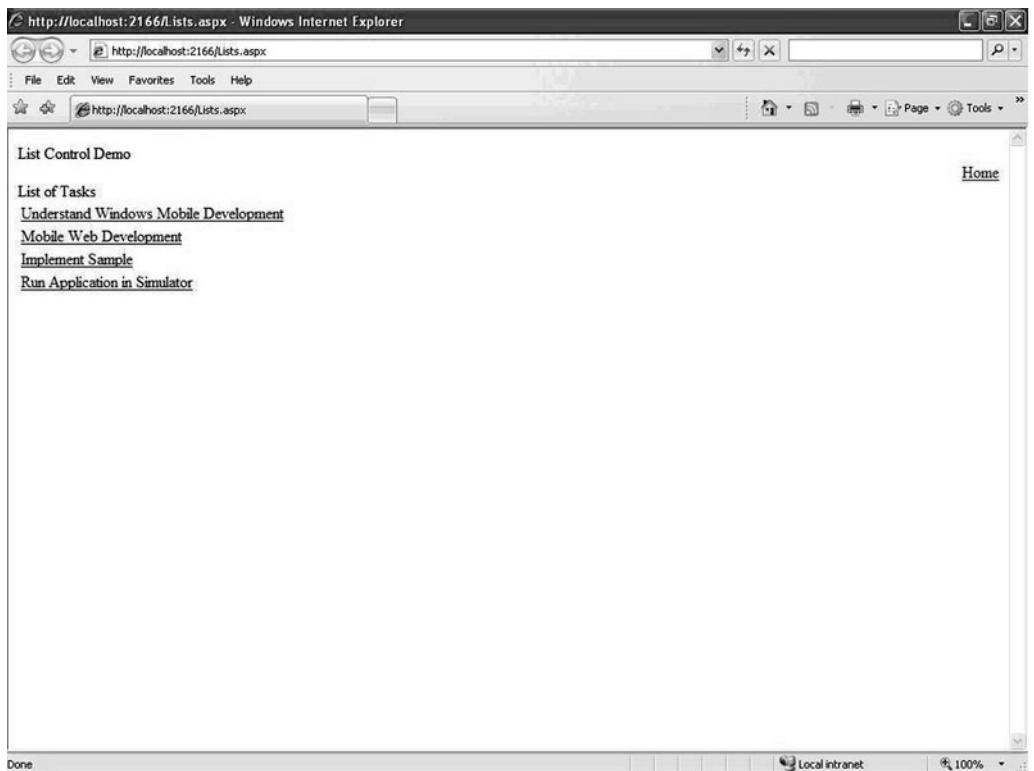
        protected void Show_TaskStatus(object sender,
ListCommandEventEventArgs e)
        {
        }
    }
}

```

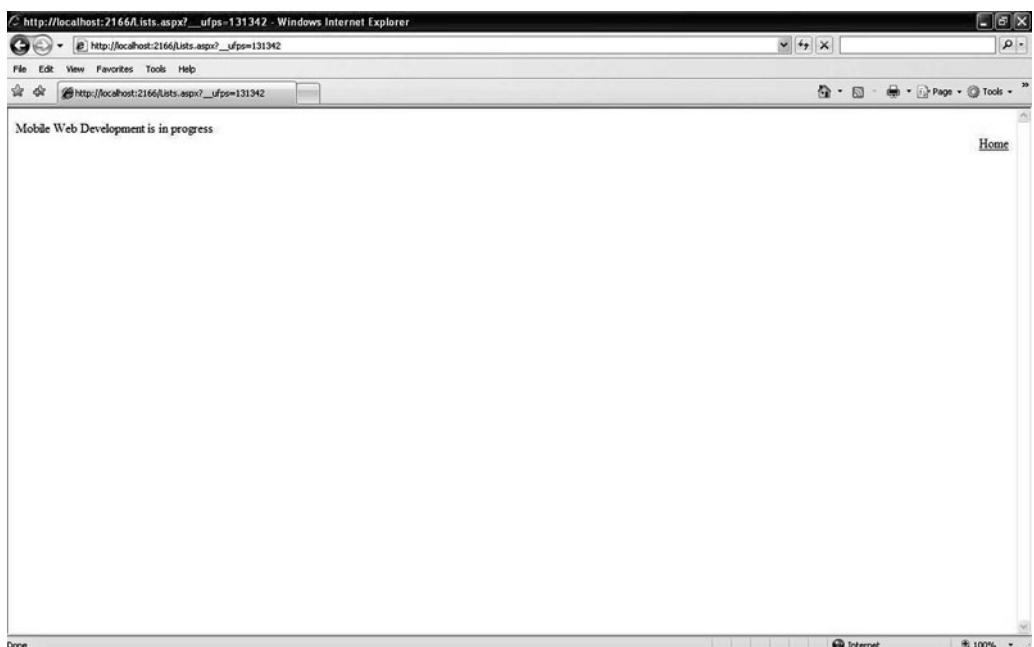
When the application runs on a mobile device, the pages will display the output, as shown in Figure 11-8 and Figure 11-9.

Figure 11-8

Output of using the List control

**Figure 11-9**

Output of using the List control



USING THE PHONECALL CONTROL

The PhoneCall control is mainly used for making a phone call to any number. The PhoneCall control contains different properties, some of which are `AlternateFormat`, `AlternateURL`, `PhoneNumber`, `SoftKeyLabel`, and `Text`. There are two different ways in which this control works:

- If a device has the ability to make phone calls, then this control presents a control that makes the phone call when triggered.
- If a device does not have the ability to make phone calls (e.g., when you run an application in a web browser), a text control or hyperlink is displayed.

The following code example demonstrates the use of the PhoneCall control:

```
<%@ Page Language="C#" AutoEventWireup="true"
Inherits="Sample.PhoneCall" Codebehind="PhoneCall.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <body>
        <mobile:Form id="Form1" runat="server">
            <mobile:Label ID="Label1" Runat="server">Dial a No.</mobile:Label>
            <mobile:PhoneCall ID="PC" Runat="server"
PhoneNumber="(800)-800-8000" Text="Help Line" AlternateFormat="{0} at {1}">
            </mobile:PhoneCall>
        </mobile:Form>
    </body>
</html>
```

In this code, the attribute `AlternateFormat` holds a value 0 and 1. Therefore, the value from the `Text` attribute is displayed. However, if the value 1 is assigned to `AlternateFormat`, the text from the `PhoneNumber` attribute will be displayed.

USING THE STYLESHEET CONTROL

The StyleSheet control defines the style used to display the controls on the mobile web page. The control itself cannot be viewed on the screen, but the defined style could be applied on the text and the controls, which is displayed on the web page. The StyleSheet control can be used to achieve a rich user interface and to maintain a consistent style throughout the page irrespective of the device. You can define the visual appearance, such as text, font, or colors across the mobile application and controls. Note that although StyleSheet controls follow the same rules of inheritance and cascading as Cascading Style Sheets (CSSs), they are not as vast as CSS. The style properties are conditional, and the style set can be applied only if the target device supports the style.

The following code sample shows how to use the StyleSheet control to apply style on a Label control:

```
<mobile:StyleSheet Runat="server" ID="lbl1">
    <mobile:Style Name="LabelStyle" Font-Name=""
Font-Bold="True" ForeColor="Red" />
</mobile:StyleSheet>

<mobile:Form id="Form1" runat="server">
    <mobile:Label StyleReference="LabelStyle" ID="Label1"
Runat="server">Welcome to Mobile World.</mobile:Label>
</mobile:Form>
```

Using this code, you created a StyleSheet control with the name `LabelStyle` and set the font style and color.

USING THE RANGEVALIDATOR CONTROL

In this section, we discuss using an input validation control, the RangeValidator.

Assume you want to create two forms. The first form should receive a number as input. On clicking the Submit button, if the number is incorrect, the second form should display an appropriate error message. You can use the RangeValidator control to validate the numbers as shown in this code:

```
<%@ Page Language="C#" AutoEventWireup="true" Inherits="Sample.
RangeValidator" Codebehind="RangeValidator.aspx.cs" %>
<%@ Register TagPrefix="mobile" Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>
<html xmlns="http://www.w3.org/1999/xhtml" >
    <body>
        <mobile:Form id="Form1" runat="server">
```

```
<mobile:Label ID="Label1" Runat="server" Text="Please enter  
a number from 1-10"></mobile:Label>  
<mobile:TextBox ID="TextBox1" Runat="server"></mobile:TextBox>  
<mobile:RangeValidator ID="rv" Runat="server"  
ControlToValidate="TextBox1" ErrorMessage="Invalid Number"  
Type = "Integer" MinimumValue="1" MaximumValue="10">  
</mobile:RangeValidator>  
<mobile:Command ID="cmd" OnClick="cmd_OnClick"  
Runat="server">Submit</mobile:Command>  
</mobile:Form>  
<mobile:Form id="Form2" runat="server">  
<mobile:Label ID="Label2" Runat="server"></mobile:Label>  
</mobile:Form>  
</body>  
</html>  
protected void cmd_OnClick(object sender, EventArgs e)  
{  
    if (Page.IsValid)  
    {  
        ActiveForm = Form2;  
        Label2.Text = "Number entered is " + TextBox1.Text;  
    }  
}
```

In this code, the first form is created with the following:

- A label with the text “Please enter a number from 1–10”
- An input box to enter a number
- A RangeValidator control that checks the range of input numbers from 1–10
- A Submit button

The second page is activated by the `OnClick` event of the Submit button on the first page, and displays a response. If the input value throws an error, an error message is displayed. Figure 11-10 and Figure 11-11 shows the output screen.

Figure 11-10

Output of using the RangeValidator Control

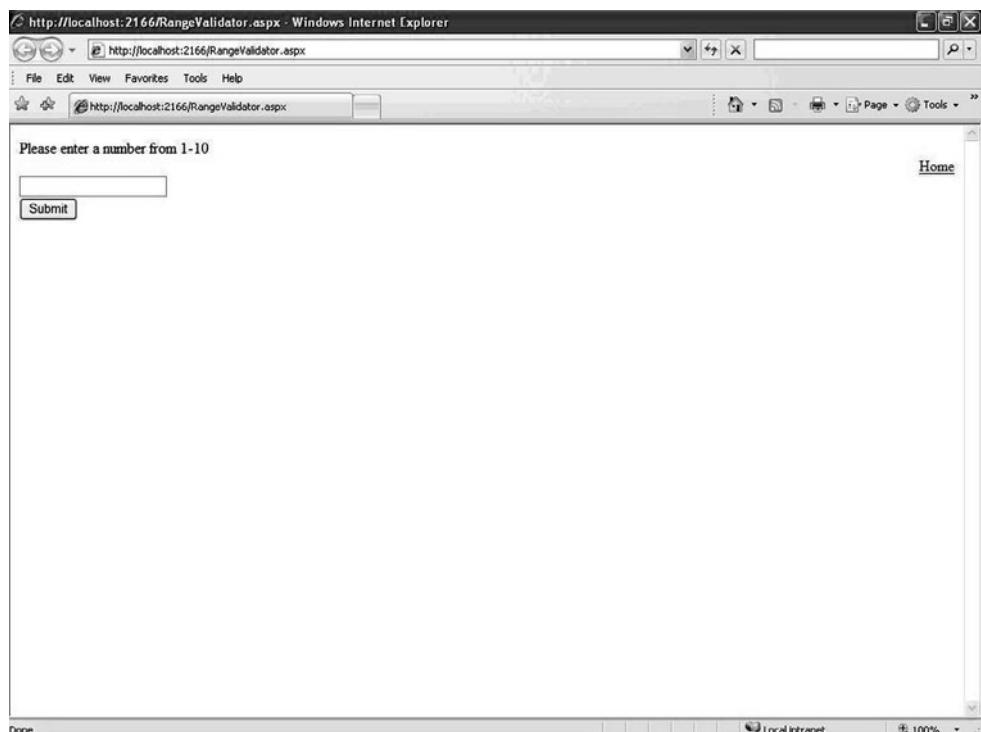
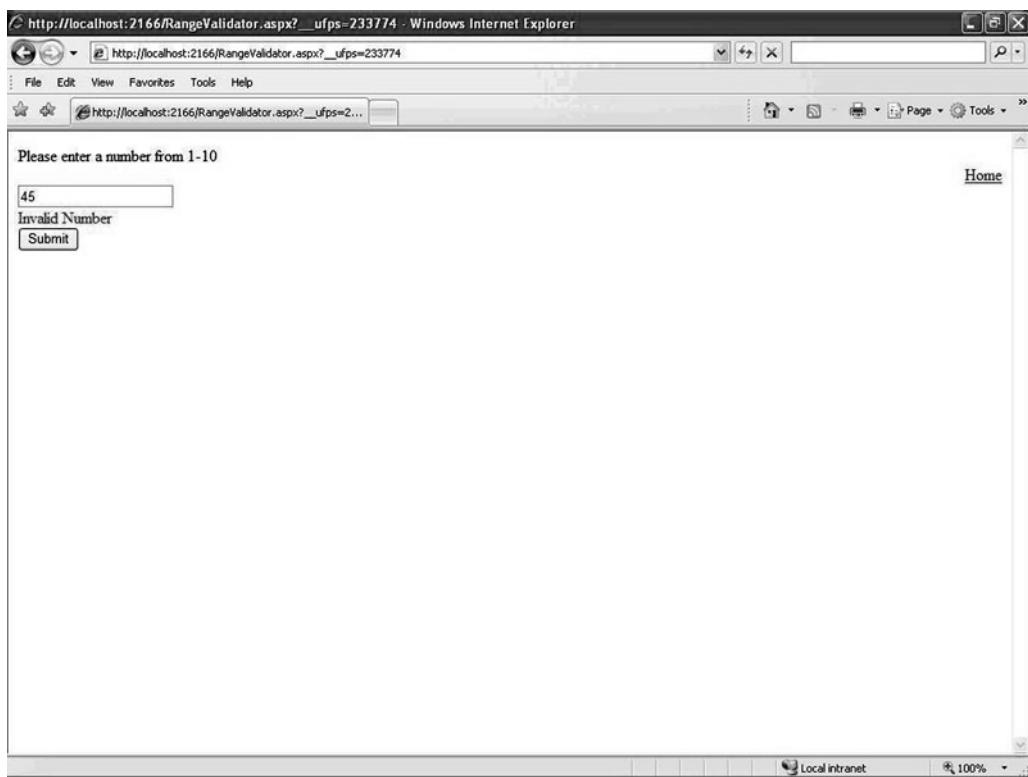


Figure 11-11

Executing the RangeValidator Control



CERTIFICATION READY?
Add mobile web controls
to a page.
6.3

Using Control Adapters

Control adapters were initially introduced in ASP.NET 2.0. Control adapters are mainly designed to render a specific set of controls to an output that is supported by the targeted mobile device.

There has been a phenomenal growth in the development of mobile devices, and these devices now support multimedia controls, online connectivity, cameras, and live broadcast. It is nearly impossible to design content with such a large variety of output without the help of control adapters to render the output so that it will be suitable for the targeted device. Multiple devices can receive the same output that is suitable and viewable without altering a single line of code. This is achievable with the help of control adapters, which render the controls so that each device can adapt them easily.

This is how the rendering process works:

1. The control is checked for the associated `Control` base class.
2. If the `Control` base class is located, then the `Render` method is applied to render the control. The `System.Web.UI.MobileControls.Adapters` namespace is used to inherit the `Control` base class.
3. If the `Control` base class is not located, then the standard `Render` method is applied to render the control.

Using control adapters, you can change the rendering of a control based on the browser type of the client. Some of the adapters that can be used with the mobile applications are:

- **ChtmlPageAdapter:** Renders ASP.NET mobile web forms pages to chtml.
- **HtmlPageAdapter:** Renders mobile web forms pages on clients to HTML.
- **WmlPageAdapter:** Renders mobile web forms pages on clients to WML.
- **XhtmlPageAdapter:** Provides a page adapter for XHTML.
- **ChtmlFormAdapter:** Provides the Form adapter for cHTML.

Next, we discuss how adapters are implemented.

In the following example, you will implement `ChtmlPageAdapter`. First, create a class named `myChtmlPageAdapter` and inherit it from `ChtmlPageAdapter`. Include the namespace `System.Web.UI.MobileControls.Adapters`. Override the `RenderForm` method of the base class `ChtmlPageAdapter` to achieve desired functionality.

The sample code snippet using the `ChtmlPageAdapter` is shown next:

```
using System;
using System.Web.Mobile;
using System.Web.UI.MobileControls;
using System.Web.UI.MobileControls.Adapters;

namespace Sample
{
    public class myChtmlPageAdapter: ChtmlPageAdapter
    {
        public override void RenderForm(HtmlMobileTextWriter writer,
Form form)
        {
            // Code to handle the ChtmlPageAdapter
        }
    }
}
```

Once you complete the class definition, then you need to use the adapter in your application. To do that, you need to add the configuration in the `web.config` file:

```
<system.web>
    <mobileControls cookielessDataDictionaryType="System.Web.Mobile.
CookielessData" allowCustomAttributes="true">
        <device name="myHtmlDeviceAdapters"
inheritsFrom="HtmlDeviceAdapters"
pageAdapter="Sample.myChtmlPageAdapter"/>
    </mobileControls>
</system.web>
```

In the `web.config` file settings, the primary attribute of the `device` element is `pageAdapter`. You need to specify the namespace along with class name so that runtime will use the `ChtmlPageAdapter`.

MORE INFORMATION

To learn more about the adapters and their usage, refer to the `System.Web.UI.MobileControls.Adapters` namespace section on MSDN.

CERTIFICATION READY?

Implement control adapters.

6.4

You can implement other adapters in a similar manner.

Understanding State Management in ASP.NET Mobile

Consider an application that you built that tracks employee data within your organization. Employees are provided access to the application based on their designation, and they can access the network using their mobile phones. There should be a mechanism to keep track of users when multiple users log on to an application. State management keeps track of what users are, what they do, when and how they logged in, and the requests that they are working on. State management is mainly responsible for handling user related information and ensuring that user state is captured appropriately.

State management can work in two different ways:

- **Client side:** Uses different methods—typically, View State, a cookie, or a `QueryString`—to capture user state. This method is a lot quicker for the user because user state information is saved on the client device.
- **Server side:** Uses the `Session` method to capture user state. The `Session` method can be resource consuming, which can cause server slow downs. If there are too many users on the server and the server is storing the information for each user state, there is the possibility that server resources may get clogged.

X REF

To understand how cookies, URLs, browser cache, and cache and session objects are used on the client side and server side respectively, refer to Lesson 7, “Programming Web Applications.”

Let's discuss some state management methods that you can use for mobile development:

- Cookieless sessions for server side
- ViewState controls for client side

IMPLEMENTING COOKIELESS SESSIONS

Using cookies is a common method for maintaining user state with the server. Every time a request from the user is initiated, the web browser uses cookie values to send information to the server. A cookie maintains state information. If the web browser is configured to accept cookies, the cookie information is stored on the web browser for processing. On the other hand, if the web browser is not capable of accepting cookies, then a cookieless session needs to be used. In this case, a new URL with a session ID is used. The session ID is used instead of the cookie.

You cannot use cookies to maintain session state in mobile applications because most mobile devices do not accept cookies. In addition, it is not advisable to use cookies in mobile applications because the use of cookies causes bandwidth overhead and is inappropriate for sensitive data. In addition, cookies can be disabled on browsers.

You can evaluate whether your mobile device supports cookies or cookieless sessions. To do this, you can use the `AutoDetect` value for the `cookieless` attribute:

```
<system.web>
<sessionState mode="InProc" cookieless="AutoDetect" />
</system.web>
```

If cookies are not available, then mobile web applications track sessions by storing the session ID in the URL as in an ASP.NET web application.

To use cookieless sessions, you need to configure the mobile application to store the session ID in the URL as in an ASP.NET web application.

To disable cookies, set the `cookieless` attribute of the `sessionState` elements to `true` in the `web.config` file.

USING VIEW STATE CONTROLS

The `ViewState` control is normally stored in the hidden fields in each page. Sending the information in the view state hidden field slows down page execution because most mobile devices transmit information very slowly. As a result, ASP.NET mobile stores view state in the session state. To disable view state for a page, set the `EnableViewState` attribute on the `@Page` directive to `false`:

```
<%@ Page Language="C#" EnableViewState="false" AutoEventWireup="true"
Inherits="Sample.RangeValidator" Codebehind="RangeValidator.aspx.cs" %>
```

View state is simple to use at a page level and can be encrypted. However, it makes a page heavy and you need to encode the view state values.

Other ways to maintain and manage state in a mobile application would be to maintain sessions and use hidden fields.

Web applications and Web sites largely use sessions for each user visit. A session life begins when the user enters the Web site or the application and ends when the user leaves. The time that the user spends in between is considered a live session. It is important to maintain consistent sessions so that the user does not have to provide information repeatedly. Mobile devices are more personal and generally not shared by the users like the computers in a cybercafé. For instance, when a user logs on to the Web site, the user should not have to repeat the login procedure until after that user logs out. You can maintain the session information by using the `System.Web.SessionState.HttpSessionState` object.

The following code sample uses the `HttpSessionState` class to store the user ID of the logged in user in the session on the `Page_Load` event:

```
using System.Web.SessionState;
protected void Page_Load(object sender, EventArgs e)
{
```

```
HttpSessionState _session = new  
HttpSessionState();  
session.Add("LoginUserID", "user1");  
}
```

CERTIFICATION READY?

Target mobile devices.

6.4

Hidden fields are present on the web form but are not visible to users, and users do not need to input any values into hidden fields. They automatically collect the information for further processing. Hidden fields can store small amounts of data and take less space. They are, like cookies, inappropriate for storing sensitive data because they can be intercepted easily in a network.

SKILL SUMMARY

In this lesson, you learned that ASP.NET enables you to develop a mobile application that serves different output devices. You saw that these mobile applications can be developed using Microsoft Visual Studio. To develop mobile web applications, you first need to create an ASP.NET Web site. Then, you need to include the System.Mobile.UI namespace and add the forms to the project. Finally, you add controls to the form and relevant code to the code-behind pages. You also learned that mobile controls always run on the server and, depending on the browser, an appropriate markup language is used to render the mobile page.

The second section of this lesson discussed a typical mobile page, which contains at least one container frame and other mobile controls such as Panels and Labels.

In the last section of this lesson, you learned about the three categories of mobile controls that are segregated based on their functions—mobile utility controls, input validation controls, and UI controls. You also saw the different elements that are used to define mobile controls. In addition, you learned that in order to support multiple mobile devices marketed by different vendors, the ASP.NET mobile web page architecture is built on the top of a device adapter model. Finally, this lesson discussed page rendering using control adapters and state management in mobile application development.

For the certification examination:

- Work with emulators and access device-specific capabilities.
- Control device-specific rendering.
- Create, view, add, and test mobile controls on mobile web applications.
- Use control adapters.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. ASP.NET enables you to configure your own device filters on the web.config file by extending the _____ class.
2. Comparison-based filters can make basic comparisons, based on a _____ argument.
3. The <Choice> elements are placed inside a _____ element.
4. _____ devices are used to develop standalone desktop applications using .NET CF.
5. You can write code to query browser-specific properties by using the _____ method.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|-------|---|
| T F | 1. Most mobile devices accept cookies. |
| T F | 2. Control adapters are mainly designed to render specific sets of controls to an output that is supported by the targeted mobile device. |

- T** **F** 3. Mobile controls extend the schema of the machine.config file.
- T** **F** 4. You can easily display mobile applications on any mobile device.
- T** **F** 5. The `System.Web.UI.MobileControls.Adapters` namespace contains the classes that can be used in the development of mobile controls.

Multiple Choice

Circle the letter that corresponds to the best answer.

1. In which of the following does ASP.NET store ViewState?
 - a. Session state
 - b. Hidden variables
 - c. Cookies
 - d. Browser cache
2. Which of the following controls validates the input against a specified pattern?
 - a. RegularExpressionValidator
 - b. RangeValidator
 - c. RequiredFieldValidator
 - d. CustomValidator
3. The outermost container in a mobile page is known as a _____ control.
 - a. Label
 - b. Form
 - c. Input Validation
 - d. Image

Review Questions

1. How does a mobile form differ from a regular ASP.NET form?
2. Why is it not advisable to use cookies in mobile applications?

■ Case Scenarios

Scenario 11-1: Displaying Product Information

A product manufacturing company wants its sales department to start using a mobile application to demonstrate the products to its clients. How will you go about developing the mobile web application?

Scenario 11-2: Developing a Mobile Web Application for Live Football Updates

You need to develop a mobile web application for football enthusiasts so that the web application will allow them to get live updates about games. What will be your approach to develop such a system?



Workplace Ready

Displaying Text on Mobile Pages

You are developing a mobile web application that allows children to read stories on mobile devices in an interactive manner. Each story has multiple lines of text. The text to be displayed is quite heavy. At the end, the children would need to reply and send the moral of the story. How will you display this?

Troubleshooting and Debugging Web Applications

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Troubleshooting web applications.	Implement tracing of a web application.	4.4
Debugging and error logging.	Configure debugging and custom errors.	4.1
Debugging and error logging.	Debug unhandled exceptions when using ASP.NET AJAX.	4.3
Debugging and error logging.	Implement the generic handler.	7.7
Debugging and error logging.	Debug deployment issues.	4.5
Troubleshooting web applications.	Implement tracing of a web application.	4.4
Debugging and error logging.	Set up an environment to perform remote debugging.	4.2

KEY TERMS

breakpoints
compilation

handlers
tracing

troubleshooting
unhandled exceptions

Developing error-free software is a challenge for many developers, but software architecture methodologies and development practices along with software libraries such as ASP.NET allow developers to minimize and remove bugs in applications through troubleshooting and then debugging.

■ Introducing Web Application Troubleshooting



Troubleshooting an application involves **tracing**. Tracing an application is the process of collecting information about web page execution. Debugging is the process of tracing an application's performance to find the sources of bugs in the source code of the application. Using tracing, you can collect information about the execution of a web page and display any diagnostic information during runtime and debugging. In Visual Studio, you also use step-through debugging where you debug each line of code. In tracing, you use methods and are able to see the trace result on a page or application level on the screen. A limitation of step-through debugging is that you need remote debugging to debug remotely deployed applications. Tracing, on the other hand, allows you to debug remotely hosted applications as well.

In ASP.NET, tracing can be configured at multiple levels. As a result, tracing can be supported even in distributed and multitier applications. ASP.NET provides extensive support to manage error pages and trap application exceptions to eliminate logical and syntactical bugs in the final product. Logical bugs might not throw any exceptions but can still be resolved with debugging and tracing. There are mainly two classes that are part of .NET Framework and can be used for any kind of .NET application, which deal with tracing and debugging—Trace and Debug. You can access these classes from the `System.Diagnostics` namespace.

Introducing Tracing

In ASP.NET, tracing offers several features. When deploying your web pages on production servers, ASP.NET allows you to add debug statements to your code directly without removing the applications from the server. Trace statements can be added before the code is deployed to the production server. Tracing also gets information from the applications already in production. Using the integrated tracing feature, you can transmit messages between the `Trace` class and the ASP.NET tracing output to analyze any problems in your application.

ASP.NET also allows you to access and modify tracing outputs, which are known as trace messages. You can analyze trace messages to understand whether the application contains errors or functions as expected. You can also use the messages to analyze application performance. As a result, you can format the trace messages per your requirements.

The `Trace` class provides several methods and properties that enable you to execute and trace the code. Here are some of the commonly used methods:

- **TraceError:** Writes the error information to trace output, which is the output generated when tracing is enabled.
- **TraceInformation:** Writes informational messages to trace output.
- **TraceWarning:** Writes warning messages to trace output.
- **Close:** Terminates the trace message listeners after flushing the output buffer.

ASP.NET provides developers with a much more refined process for displaying page-level information. With ASP.NET, rather than using `Response.Write`, use `Trace.Write` or `Trace.Warn`. Both methods accept two parameters each, as shown:

- **Trace.Write (category, message):** This displays informational messages. Individual operations and messages, such as `Begin PreInit` will display informational messages.
- **Trace.Warn (category, message):** This displays text warnings in a red color for highlighting the warnings. `ArgumentException` and `InvalidOperationException` information can be a type of warning.

Practically, tracing can be done at two levels. Tracing an individual web page is called page tracing, and enabling tracing for the entire application is called application tracing. Application tracing in ASP.NET allows you to view the most recent tracing data without restarting a tracing session.

UNDERSTANDING PAGE AND APPLICATION TRACING

When your application generates an error, you need to start the debugging process. However, before you do this, you need to review ASP.NET page tracing. A page is made up of several server-side controls organized in a hierarchy. These controls in turn may have other controls nested within them. Tracing helps you understand the flow and hierarchy of the controls in a page and in turn allows for easy debugging of the application.

Application tracing means that the entire application will be traced for any execution issues. This tracing provides the same result as page tracing; however, the trace contextual information will not be displayed at the end of each HTML page. Instead, a new page, `Trace.axd`, containing all the related information for the entire application is displayed with the help of a special handler. This `Trace.axd` file is created in the application root folder and can be easily viewed in the browser by navigating to the application root folder \`Trace.axd`. The advantage of

application tracing over page tracing is that page tracing adds a large amount of trace data to the page output. As a result, developers have to go through the entire list to identify the bugs. However, if application tracing is implemented, developers can view all requested pages in the application on a single output page and can refer to further details later when identifying bugs.

ENABLING TRACING

To use tracing, you need to use the `Trace` class, which contains methods, such as `Trace.Write` and `Trace.Warn` that you can use to enable tracing. This class is part of the `System.Diagnostics` namespace. You use the `Trace` class in the code behind.

The `Trace` element configures the ASP.NET code tracing service that controls how trace results are gathered, stored, and displayed. It is generally configured on the `web.config` file. To do so, use the following code:

```
<system.web>
<trace enabled="true" pageOutput="true" requestLimit="10"
mostRecent="true" traceMode="SortByTime" localOnly="true"/>
</system.web>
```

Table 12-1 lists some properties of the `Trace` element that can be manipulated programmatically.

Table 12-1

Trace element properties

ATTRIBUTES	DESCRIPTION
Enabled	Enables or disables application-level tracing. The default value is <code>false</code> .
localOnly	Specifies whether the trace output should be restricted to the local computer. When set to <code>false</code> , this property enables remote viewing of the trace output.
pageOutput	Specifies whether the trace output should be displayed on individual pages, in addition to caching application-level messages in an individual web page.
requestLimit	Specifies how many trace messages can be stored before removing the earlier set of trace messages.
traceMode	Gets or sets the sorted order in which trace messages should be output to a requesting browser.

TAKE NOTE *

Consider a case where you have enabled Trace at the application level. For a particular page in the application, you can set the `Trace` attribute of the `@Page` directive to `false` for that page, the Trace output will not be shown. The application-level tracing configuration can be overridden by setting the trace attributes at the page level. The following sections discuss how to enable page-level tracing and application-level tracing.

To turn page tracing on, you need to set the `Trace` property of the `Page` class to `true`. You can also define the order of the trace output by using the `TraceMode` attribute of the `Trace` element in the `web.config` file. These activities can be done either in the source `.aspx` file of the page or by using the Visual Studio Designer.

The code-behind file shows you how to enable tracing for a page using the `@Page` directive in HTML and also by using the `Trace` class in the code-behind file.

Create a new web application. Add a new web page in the solution and name it `PageTrace.aspx`. Copy the provided code in the `Trace.aspx` page:

```
<%@ Page Language="C#" AutoEventWireup="true"
Trace="true" TraceMode="SortByTime" CodeBehind="PageTrace.aspx.cs"
Inherits="Trace" %>
```

The code that will go in the code-behind file is:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Enable Page Level Tracing
    Trace.IsEnabled = true;
    // Set Trace Mode

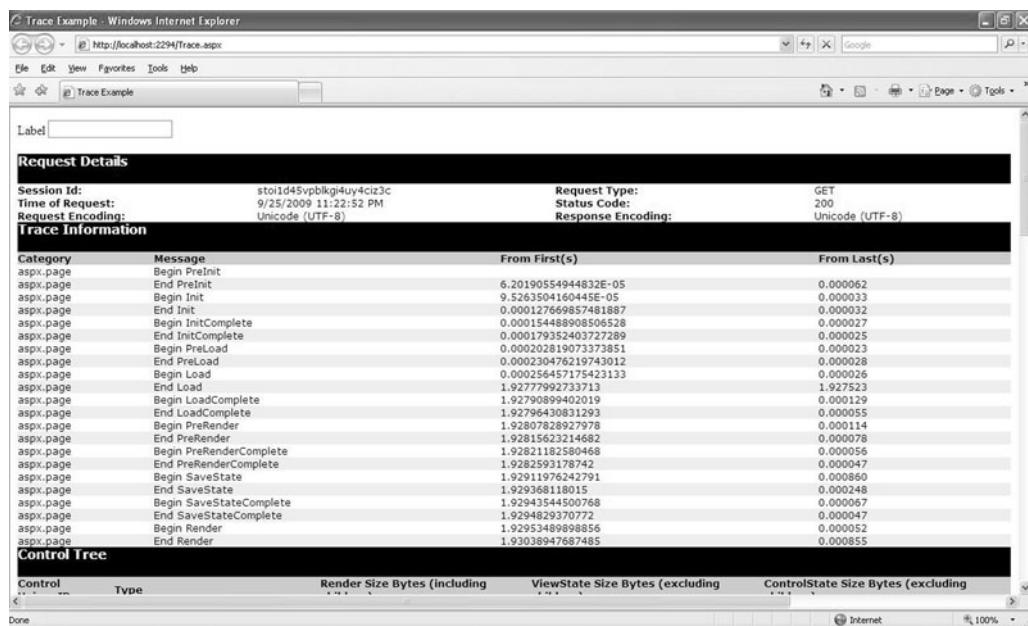
    Trace.TraceMode = TraceMode.SortByTime;

    // To disable page-level tracing, use the following code:
    // Trace.IsEnabled = false;
}
```

In this code, you have enabled the `Trace` property on the `Trace.aspx.cs` page. When the `Trace` property is set to `true`, the tracing information will be displayed at the end of the HTML code as shown in Figure 12-1.

Figure 12-1

Displaying tracing information



TAKE NOTE*

Page tracing generates a lot of information about the background processes, some of which can be extraneous. Developers need to sort the information to identify the root cause of a problem. ASP.NET provides the `System.Diagnostics` namespace to manage tracing and debugging information. To include this namespace in your program, just insert a line of code in the code-behind file and use the `Trace` class to display the trace information.

You can turn application tracing on by setting the `enabled` property of the `Trace` element in the `web.config` file to `true`.

IMPLEMENT APPLICATION TRACING

TAKE NOTE*

To use application-level tracing, make sure to set the `Trace` property in the `@Page` directive to `false` to ensure that page-level tracing is turned off.

1. Open `PageTrace.aspx` that you created earlier.
2. To enable application-level tracing, include the `Trace` element in the `web.config` file:


```
<system.Web>
  <!--
  -->
  <trace enabled="true" pageOutput="true" requestLimit="10"
mostRecent="true" traceMode="SortByTime" localOnly="true" />
</system.Web>
```

This code enables tracing, sets the maximum number of tracing messages to 10, and disables remote viewing of trace messages. If you don't want to see the tracing information on each individual page, then set `pageOutput = false`.

3. View the `Trace.aspx` page in a browser to verify whether the application-level trace output is shown at the page level.
4. To view application tracing, .NET provides a special handler, `Trace.axd`. To open `Trace.axd` in the address bar of the browser, replace `PageTrace.aspx` with `Trace.axd`. `Trace.axd` is a built-in trace viewer that displays the trace information for every page in an ASP.NET application. It will display the output as shown in Figure 12-2.

Figure 12-2

Tracing output

The screenshot shows a Microsoft Internet Explorer window with the title bar "http://localhost:2294/Trace.axd - Windows Internet Explorer". The address bar also contains "http://localhost:2294/Trace.axd". The page content is titled "Application Trace" with a link "[clear current trace]" and a note "Physical Directory:C:\Sample\Debug\Debug\". Below this is a table titled "Requests to this Application" with columns: No., Time of Request, File, Status Code, Verb, and Remaining: 1. The table lists 9 requests from 8/6/2009 at 5:37:07 AM to 5:38:30 AM, all for "Trace.aspx" with status 200 and verb GET, each with a "View Details" link. At the bottom of the page is the footer "Microsoft .NET Framework Version:2.0.50727.1433; ASP.NET Version:2.0.50727.1433".

TAKE NOTE *

ASP.NET displays individual page-level trace information at the end of a page in an organized manner. Some of the parameters that the output is organized by are Request Details, Trace Information, Control Tree, Session State, Application State, and Cookies Collection variables.

You have seen how you can display trace output in a browser programmatically. In addition to this, ASP.NET provides you with the capability to generate and store trace messages to an external file. This aspect is useful when you want to log the compiler output of an application.

ADDING TRACE STATEMENTS

The tracing information displayed in the output HTML page includes the specific trace statements printed during the execution of the page. You can use these statements to monitor the progress of page execution. For example, to verify whether a page is loaded and initialized properly, you can print customized messages at every stage of page initialization, such as load, load complete, and pre-render stages.



ADD CUSTOM TRACE STATEMENTS

To add custom trace statements:

1. Create a new Web site, and insert a new web page called Trace.aspx.
2. Enable tracing by setting the Trace property in the @Page directive to true.
3. Run the page to ensure that page tracing is occurring.
4. Open the Trace.aspx.cs file.
5. Add trace statements in the event methods that will be executed during the initialization of the page:

```
protected void Page_Load(object sender, EventArgs e)
{
    Trace.Write("Tracing Page Load...");
}

protected void Page_LoadComplete(object sender, EventArgs e)
{
    Trace.Write("Tracing Page Load Complete...");
}

protected override void OnPreRender(EventArgs e)
{
    Trace.Write("Tracing Page OnPreRender...");
}
```

6. Compile the program, and run the Web site. The trace statements will appear in the output HTML, as shown in Figure 12-3.

Figure 12-3

Trace statements

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit	5.25206415899227E-05	0.000053
aspx.page	End PreInit	8.63238204855645E-05	0.000034
aspx.page	Begin Init	0.0001167746180003126	0.000030
aspx.page	End Init	0.000141358748109047	0.000025
aspx.page	Begin InitComplete	0.000165104782870449	0.000024
aspx.page	End InitComplete	0.000188012722287733	0.000023
aspx.page	Begin PreLoad	0.00021147933220393891	0.000023
aspx.page	End PreLoad	0.000234208122039812	0.000023
aspx.page	Begin Load...	5.36783194512152	5.367598
aspx.page	End Load	5.36797861180681	0.000147
aspx.page	Begin LoadComplete	5.36804286578322	0.000064
aspx.page	Tracing Page Load Complete...	5.3680951070597	0.000052
aspx.page	End LoadComplete	5.36814287849433	0.000048
aspx.page	Begin PreRender	5.36818869437317	0.000046
aspx.page	Tracing Page OnPreRender...	5.36823618644269	0.000047
aspx.page	End PreRender	5.36829624994238	0.000060
aspx.page	Begin PreRenderComplete	5.36835156423512	0.000055
aspx.page	End PreRenderComplete	5.36841559840574	0.000055
aspx.page	Begin SaveState	5.36894045405980	0.000040
aspx.page	End SaveState	5.36894087237344	0.000094
aspx.page	Begin SaveStateComplete	5.37002887238462	0.000088
aspx.page	End SaveStateComplete	5.37013531049337	0.000106
aspx.page	Begin Render	5.37018783113496	0.000053
aspx.page	End Render	5.37088037725465	0.000693

USING THE TRACEFINISHED EVENT

ASP.NET allows you to log the application- or page-level trace information generated for any troubleshooting purposes. You can achieve this with the help of the `TraceFinished` event of the `TraceContext` class. This event is triggered after all the requested information is gathered using the `OnTraceFinished` method. If you don't need the trace information, you can choose to discard the trace output. You may need to discard older versions of trace data to store latest versions or even backtrack to older versions of trace data. To use the `TraceFinished` event, you need to first register a handler for the event in the `Page_Load` method or `Page_Init` method.

The following code illustrates how to code the `OnTraceFinished` event:

```
protected void Page_Load(object sender, EventArgs e)
{
    // Register a handler for the TraceFinished event.
    Trace.TraceFinished += new TraceContextEventHandler(this.
OnTraceFinished);
}
protected void OnTraceFinished(object sender,
TraceContextEventArgs e)
{
    TraceContextRecord TCR = null;
    // Traverse through the collection of trace records and
    write it to Response.
    foreach (object obj in e.TraceRecords)
    {
        TCR = (TraceContextRecord)obj;
        Response.Write("Trace Message: " + TCR.Message + "<BR>");
    }
}
```

This code registers the `TraceFinished` event of the `TraceContext` class in the `Page_Load` event of the web page. Then, it handles the `TraceFinished` event with the `OnTraceFinished` method. In this method, you create an object of the `TraceContextRecord` class that represents an ASP.NET trace message and information associated with the trace. Then it traverses through all records of the Trace collection known as `TraceRecords`, which is an object that encapsulates the trace message, category, and associated exception and writes them to the `response` object.

The following code example illustrates how to programmatically control tracing at runtime based on specific user input:

```
if (!String.IsNullOrEmpty(Request.QueryString["UID"]))
{
    if (Request.QueryString["UID"] == "100")
    {
        Trace.IsEnabled = true;
    }
    else
    {
        Trace.IsEnabled = false;
    }
}
```

This code enables tracing if the ID of the user querying is 100. Alternatively, you can write the following code to control tracing at runtime:

```
this.Trace.IsEnabled = this.Request.QueryString["UID"] == "100";
```

When you execute the previous code snippets, you will get the output similar to Figure 12-3.

CERTIFICATION READY?

Implement tracing of a web application.

4.4

■ Introducing Debugging and Error Logging



THE BOTTOM LINE

When testing an ASP.NET application, several types of errors can occur. **Compilation** and syntax errors can be eliminated during the compilation stage itself. However, other categories of errors can be fixed or resolved only by debugging the code. Visual Studio provides extensive and efficient capabilities to debug an application and log the errors reported.

When developing an application, it is common practice to insert tracing messages into the web pages and run them to view the trace output. You can examine the trace output to verify the desired working of your application. Though this technique is helpful, debugging is always preferred because instead of reviewing the entire trace output, the developer can utilize the time to fix the problems in the code.

To enable debugging of an application, you need to set the application build mode to Debug. You can do this using the compilation element in the application's web.config file, as shown in this code example:

```
<system.Web>
  <compilation debug="true">
</system.Web>
```

TAKE NOTE*

In this code, when you set the debug option to `true`, it may affect performance. Therefore, set this value to `true` only during development.

X REF

Refer to the section on remote debugging later in this lesson to understand why it is needed.

Application debugging can be performed on a local computer or from a remote location. Local debugging can be done for applications on the local machine using ASP.NET Development Server provided by VS.NET. However, for remote debugging, you need to use the remote debugging approach.

TAKE NOTE*

An application that has been configured for a debug build will execute slowly when compared to an application configured for a release build. Therefore, it is recommended that you turn debugging off before deploying your application.

TAKE NOTE*

If you press F5 to execute the code without enabling debug, it will prompt you to choose whether you want to enable it. On confirming, the web.config value will be set automatically.

Debugging with Visual Studio

Visual Studio provides excellent debugging features in its development environment. In Visual Studio, you can debug all the steps of code execution with the help of watch windows. You can also pause the automatic code execution and step through the execution of code line by line. All the debugging features available for desktop applications are available for an ASP.NET application also.

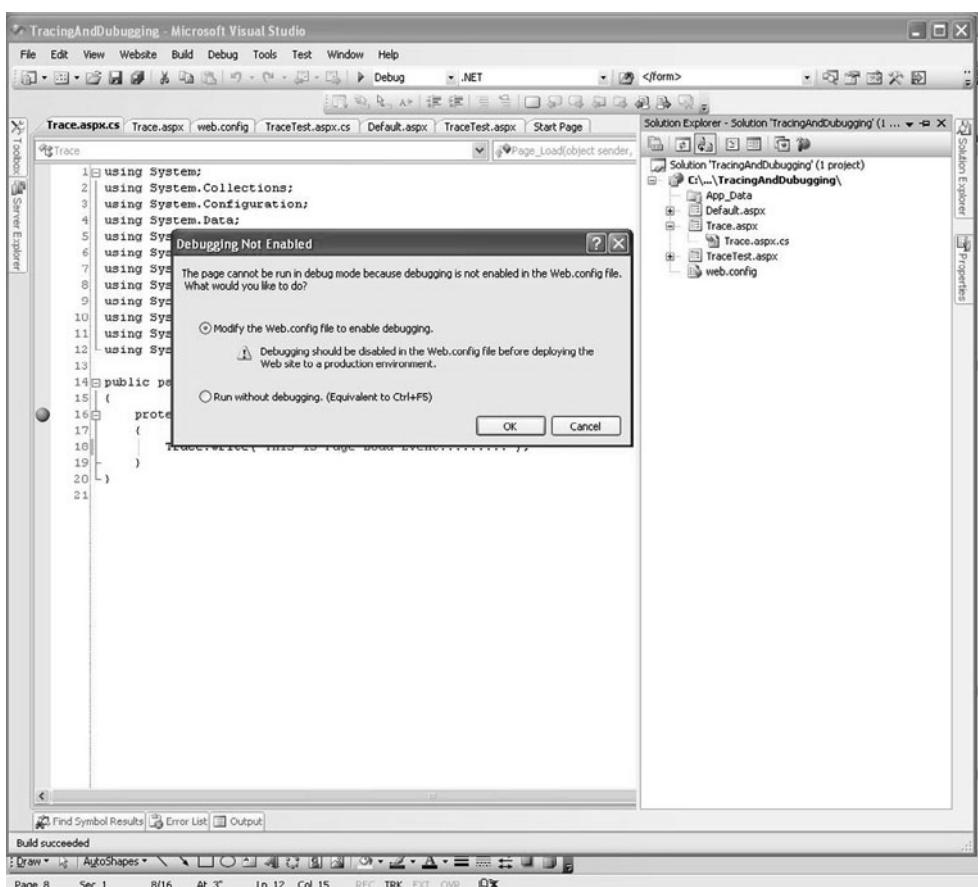
Consider that you need to debug your Web site. To support debugging, Web.config needs to include the appropriate settings. Although, you can type the debugger setting, Visual Studio automatically generates the code for you once you start debugging:

```
<system.Web>
  <trace enabled="true" />
  <compilation debug="true">
</ system.Web>
```

Figure 12-4 shows the dialog box with the message that prompts you to run without debugging or modify the web.config file to debug.

Figure 12-4

Enabling debugging using web.config



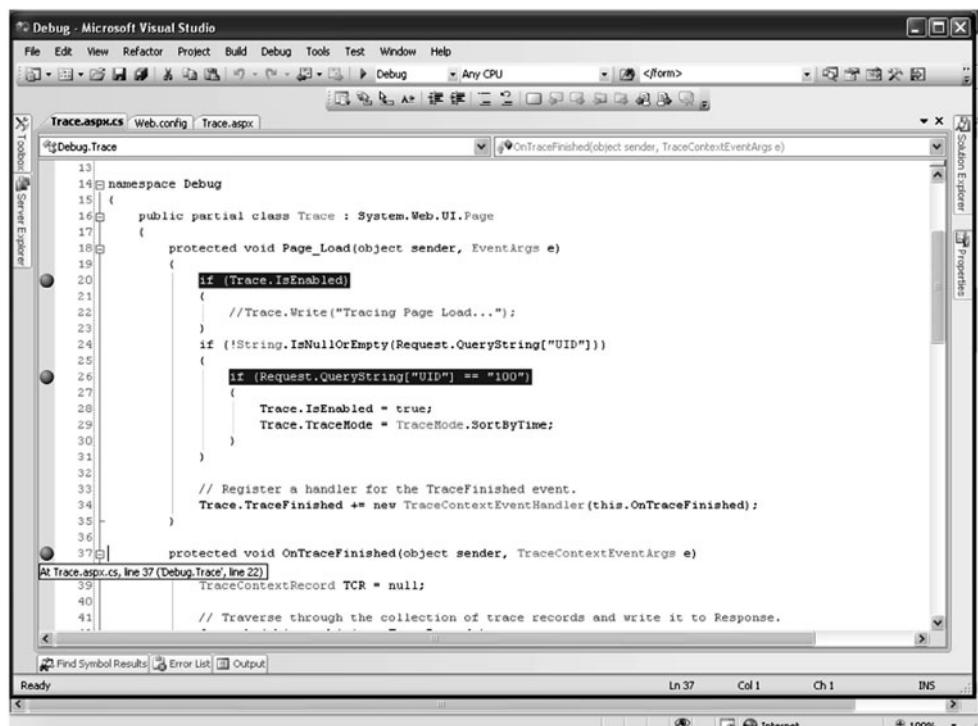
DEBUG AN APPLICATION

To debug the application:

1. Add the Trace.aspx page.
2. Insert breakpoints in Page_Load as shown in Figure 12-5.

Figure 12-5

Inserting breakpoints



TAKE NOTE *

You can insert **breakpoints** by highlighting a line in the editor window and pressing the F9 key. Another way would be to use the Toggle Breakpoint option from the Debug menu. However, the simplest way is to click the margin in Visual Studio editor in front of the line where you want to put the breakpoint.

3. Press the F5 key to start debugging.

TAKE NOTE *

You can also debug by selecting the Start Debugging option from the Debug menu. Note that you need to turn on debugging in the web.config file. While debugging, when it comes to your breakpoints, Visual Studio will pause execution and highlight the current line in yellow in the window.

In this example, Page_Load is the first breakpoint Visual Studio encounters. Pressing F10 steps over methods, whereas pressing F11 steps into methods. Alternatively, you may use Debug, Step Over and Debug, Step Into from the main menu.

4. Place your mouse cursor over the variables to see the value that appears on the ToolTip.
5. Press F5 to resume the program. Visual Studio will run until it hits another breakpoint. Run through all the breakpoints.
6. Select Refresh Page to post the page back to the server. Note that the breakpoints are hit again. A new page is created for each request that comes in.

TAKE NOTE *

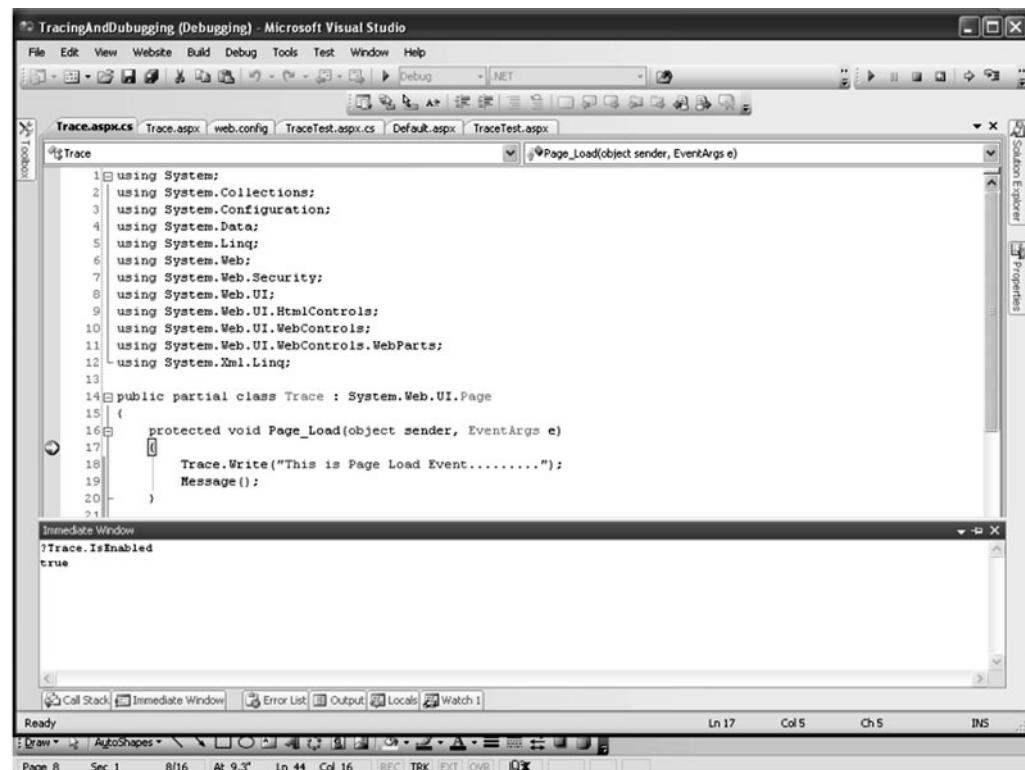
While debugging, you can use the Step Into feature to go through the code or go into the next function or loop. By using Step Over, you can choose to skip going into the function or loop. If you just execute the code while using Step Out, you can skip a function without debugging it.

Visual Studio provides some windows for monitoring various aspects of an application:

- **Immediate window:** Allows you to enter expressions to be evaluated or executed by the development language during debugging, as shown in Figure 12-6.

Figure 12-6

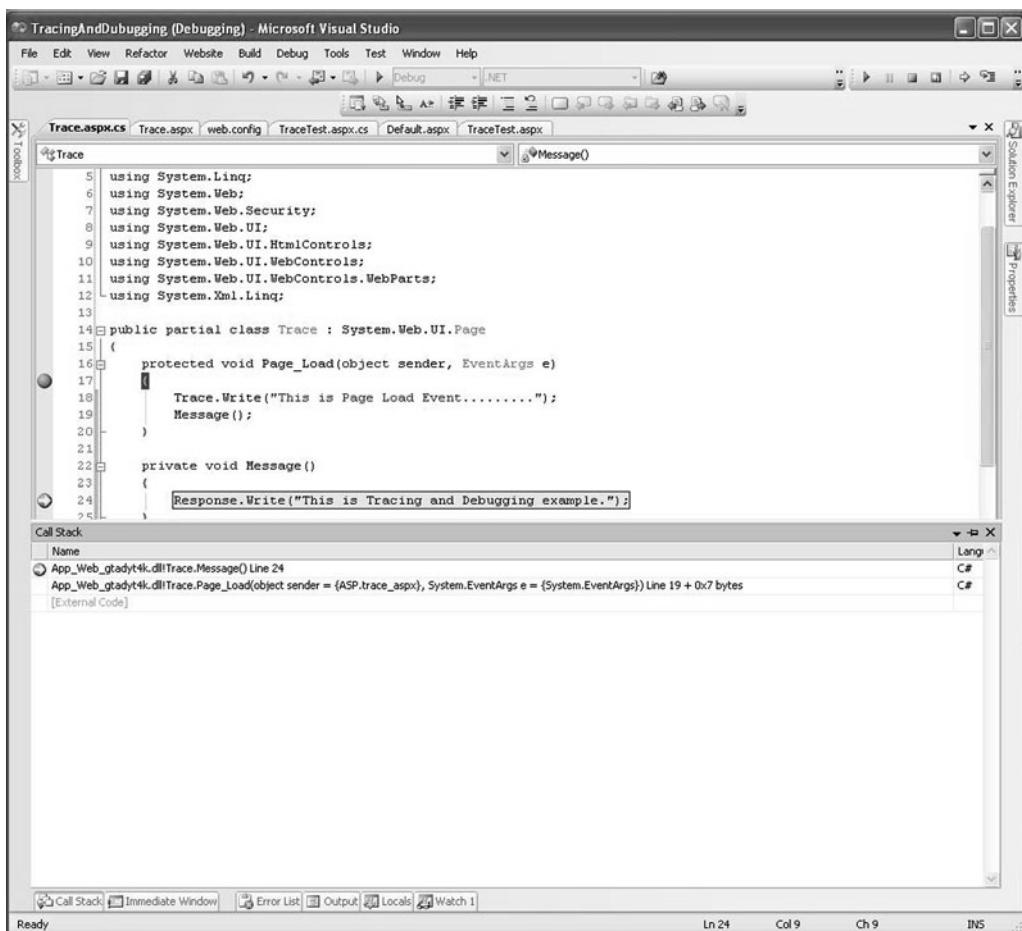
Immediate window



- **Call Stack window:** Allows you to monitor the execution and see the values of variables or controls as shown in Figure 12-7. The Call Stack window displays the sequence in which the methods are called. It helps in backtracking when a problem occurs in the sequence of methods.

Figure 12-7

Call Stack window



- **Watch window:** Displays the values of the variables generated. This helps in monitoring the final output step by step.
- **Local window:** Displays information about the variables and controls within a method.
- **Auto window:** Displays information about the variables, properties, and constants used in the current line of code and three lines prior to and post the current line of code. Although this is similar to the function of the Local window, the information in the Local window is more elaborate.

Debugging Deployment Issues

After you develop an application, you may encounter some errors while deploying it. Most of the errors are due to deployment issues in the IIS Web Server. Errors may be thrown if IIS is installed after the .NET Framework installation or if any file extension is deleted from the IIS system. Here is one common error that you may encounter:

"Unable to start debugging on the web server. The underlying connection was closed: An unexpected error occurred on a receive."

To avoid such errors associated with IIS, ensure that your web application uses integrated Windows authentication. In the web.config file, you need to specify `<compilation debug="true">`. You also need to make sure you install IIS before installing Visual Studio.



Refer to Lesson 1
“Introducing ASP.NET 3.5” for more information on IIS and creating an IIS web application.

+ MORE INFORMATION

To know more about the options that you can use with the aspnet_regiis.exe tool, refer to the ASP.NET IIS Registration Tool section on MSDN.

X REF

Refer to Lesson 13 “Deploying and Publishing Web Applications” to know more about setting .NET Framework versions.

CERTIFICATION READY?

Debug deployment issues.

4.5

Another issue that you may encounter in large applications is the creation of multiple assemblies while deploying, which you may overwrite or use. You can use the aspnet_merge.exe tool to merge and manage the output assemblies and files that are deployed on the server.

+ MORE INFORMATION

For more information on the aspnet_merge.exe tool, refer to the ASP.NET Merge Tool section on MSDN.

Debugging Remotely

Remote debugging is the process of debugging an application on a remote system where you control the application from another system. Visual Studio 2008 provides the remote debugging feature.

To use this functionality, you need to install Visual Studio Remote Debugging Monitor (Mvsmon.exe) on the remote machine and enable administrative privileges on your computer. The system that houses the application and where debugging has to be carried out is referred to as the remote system. The system that you want to debug from is referred to as the local system.

Remote debugging is specifically important in certain scenarios. Sometimes, you may not be able to access an application or run a page locally. In such scenarios, you may need to use remote debugging.

CERTIFICATION READY?

Set up an environment to perform remote debugging.

4.2

In applications using graphical user interface (GUI), you will find it easier to keep application and debugger interactions separate. This way, you will be able to separate your interactions with the application on the remote server from the interactions with the debugger on the local server. As such, remote debugging is very useful. Using remote debugging, you can control the application and take appropriate steps to rectify errors.

+ MORE INFORMATION

The Mvsmon.exe is a small application that servers as the remote debugging monitor installer. For more information on remote debugging, refer to the How to: Set Up Remote Debugging section on MSDN.

Using Error Handling

The errors that occur when executing an application can be handled at the page level or application level. The common approach to handle errors at the page level is to include try/catch blocks in the source code.

TAKE NOTE *

You may use the built-in support provided by Visual Studio to handle exceptions that may occur when executing applications. To handle the exceptions, use the try-catch keywords that consist of a try block and one or more catch blocks, which contain code for handling specific exceptions. For more information, refer to “How to: Use the Try/Catch Block to Catch Exceptions” on MSDN.

The following lines of code show the syntax of a try/catch block:

```
try
{
    //
}
catch
{
    // Catch Exception
}
```

An alternate method is to add code to the `Page_Error` handler so that when an error occurs, the user is redirected to an appropriate error page that will provide a detailed description of the error. The following code example illustrates such a redirection:

```
protected void Page_Error(object sender, EventArgs e)
{
    Exception ex = Server.GetLastError();
    if (ex is System.IO.FileNotFoundException)
    {
        Response.Redirect("FileNotFoundException.aspx");
    }
}
```

This code example redirects the control to the `FileNotFoundException.aspx` page when a file requested by the application is not found. You can use the `Global.asax` file to handle application-level errors. The errors are usually logged in the event log, text file, or database.

Note that you cannot use `response.redirect` from inside a catch block because this is a threading-related problem. You would need to provide a Boolean workaround as shown:

```
bool needredirect = false;
try
{
    ...
}
catch
{
    needredirect = true;
}
if (needredirect)
    Response.Redirect("...");
```

TAKE NOTE *

CUSTOMIZING ERROR PAGES

Instead of displaying the standard ASP.NET error messages, such as `FileNotFoundException`, `PageInitialization`, and `PageLoad`, you can also display customized error pages that will display the messages that you would like the user to see when a particular error occurs. The information about the error page to be displayed needs to be provided in the `web.config` file of the application. Table 12-2 lists the configuration settings of custom error pages.

Table 12-2

Configuration settings of custom error pages

ATTRIBUTE	DESCRIPTION
<code>defaultRedirect</code>	Directs the user to the specified URL in the event of an exception.
<code>Mode</code>	Specifies whether the server returns partial or complete exception information to local and remote callers. The default value is <code>RemoteOnly</code> , which returns complete exception information to callers on the same computer as the server.



DISPLAY A CUSTOM ERROR PAGE

To display a custom error page in ASP.NET:

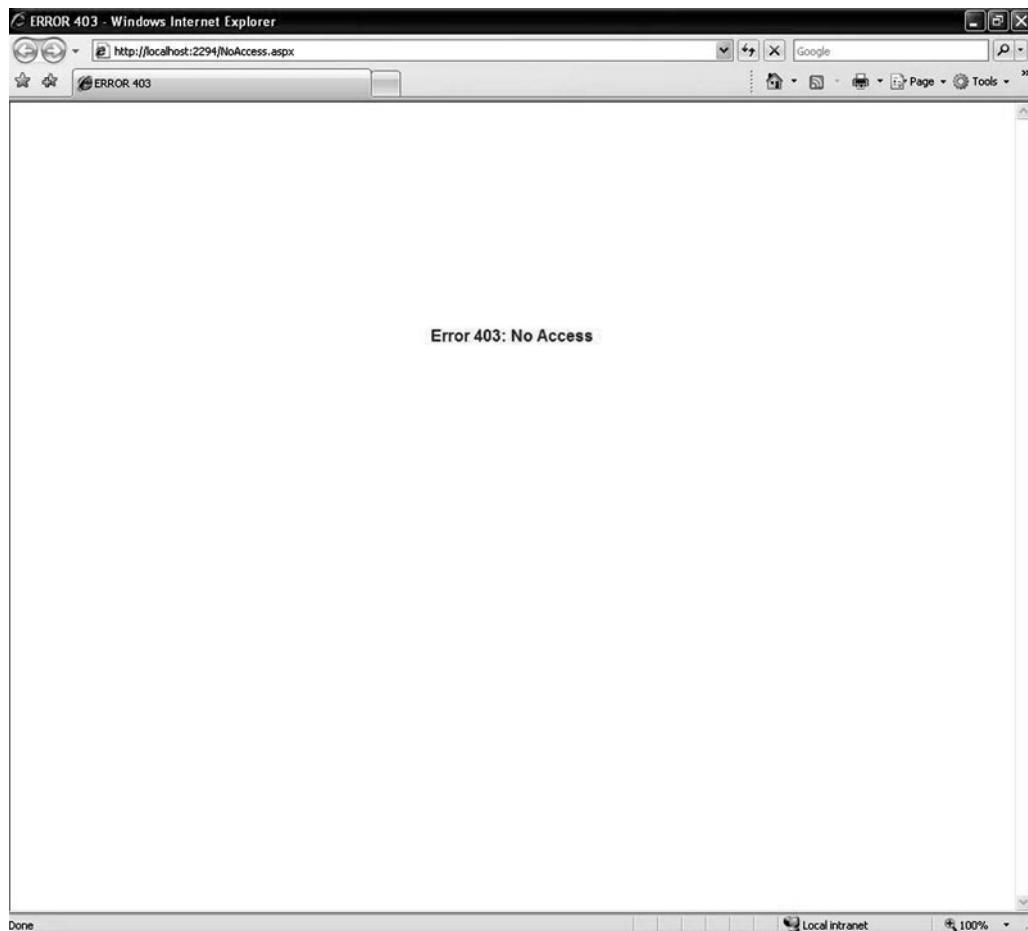
TAKE NOTE *

The code samples provided here are IIS 6 specific. IIS 7 handles this in a different way. For more information refer to the Edit a Custom HTTP Error Response section on MSDN.

1. Open a Web project.
2. Add a new web form named "ThrowErrors.aspx" to the solution.
3. To the form, add two buttons, one to throw a 404 error, which is the standard error number for the File Not Found error, and the other to display the error page.
4. Add two ASPX pages that will act as custom error pages, one as "FileNotFoundException.aspx" and the other as "NoAccess.aspx."
5. Add messages to the custom error pages as shown in Figure 12-8.

Figure 12-8

Adding messages to custom error pages



6. Add the following lines of code to the web.config file so that custom error pages are called when the appropriate error occurs:

```
<system.Web>
    <customErrors mode="RemoteOnly"
defaultRedirect="ErrorPage.aspx">
        <error statusCode="403" redirect="NoAccess.aspx" />
        <error statusCode="404" redirect="FileNotFoundException.aspx" />
    </customErrors>
<system.Web>
```

7. In the main form of your application, add event handlers for the button clicks so that the corresponding custom error pages are displayed. The following code example illustrates that if the user is redirected to a NonExistent.aspx page, which is missing in the application, it will throw a File Not Found exception.

```
protected void btnThrow404_Click(object sender, EventArgs e)
{
    try
    {
        // NonExistent.aspx is not present in the project, so
        // it will throw a File not found exception.
        Response.Redirect("NonExistent.aspx");
    }
    catch (Exception ex)
    {
        throw new System.IO.FileNotFoundException("File Not
        Found", ex);
    }
}
```

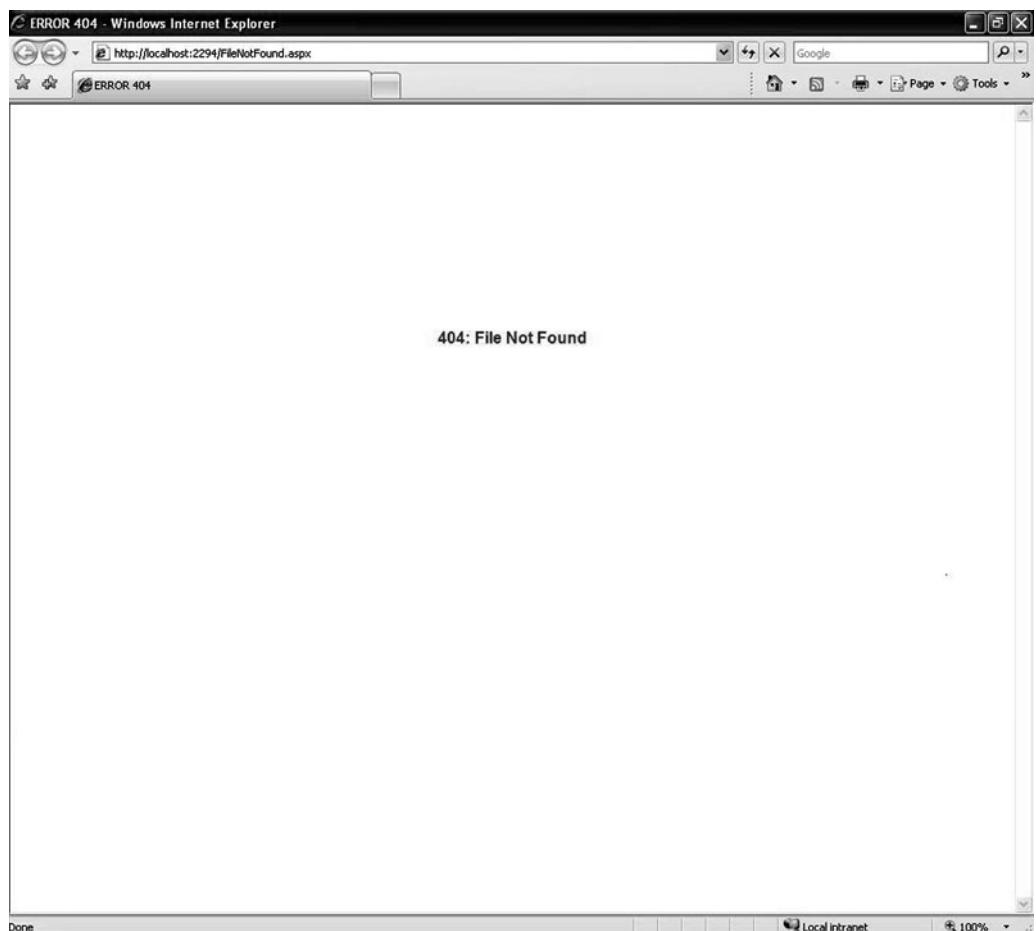
CERTIFICATION READY?

Configure debugging and
custom errors.
4.1

If you run this example in the debugger, the debugger will break as soon as an exception is encountered, as shown in Figure 12-9. To continue executing the code and display the error page after Visual Studio reports the exception, press F5.

Figure 12-9

File not found error page



UNDERSTANDING UNHANDLED EXCEPTIONS

Unhandled exceptions are generated by an application when it encounters an unexpected error during development time. One common way to handle these exceptions in ASP.NET is to redirect the control to the default error page. However, this error page cannot provide all the necessary information about the exception to the end user. To overcome this, ASP.NET allows you to trap exceptions by setting up an error handler. This handler is activated when error events are triggered in the `HttpApplication`.

To implement this approach, you need to define a handler in the `HttpApplication`-derived class within the `Global.asax` file. Now, whenever exceptions occur, your application will start receiving notifications because the handler is connected to the `Application_Error` event. This can help you log the error or show it on the debug console before redirecting the user to an error page.



HANDLE AN UNHANDLED EXCEPTION

To handle `HttpApplication` exceptions:

1. Open your ASP.NET application in Visual Studio.
2. Create a new page named `Debug.aspx` and open its code-behind file `Debug.aspx.cs`.
3. Add the following lines of code to throw an `ArgumentOutOfRangeException` in the `Page_Load` event:

```
protected void Page_Load(object sender, EventArgs e)
{
    throw new ArgumentOutOfRangeException();
}
```

4. Open the `Global.asax` file in your project. This file contains all the different event handlers. Add the following lines of code to the `Application_Error` handler:

```
protected void Application_Error(object sender, EventArgs e)
{
    Exception objExp = Server.GetLastError().
GetBaseException();

    String strError = "Error in: " +
Request.Url.ToString() + "\n" + "Error Message: " +
objExp.Message.ToString() + "\n" + "Stack Trace:" +
objExp.StackTrace.ToString();

    EventLog.WriteEntry("Trace_WebApp", strError,
EventLogEntryType.Error);
    Server.ClearError();
}
```

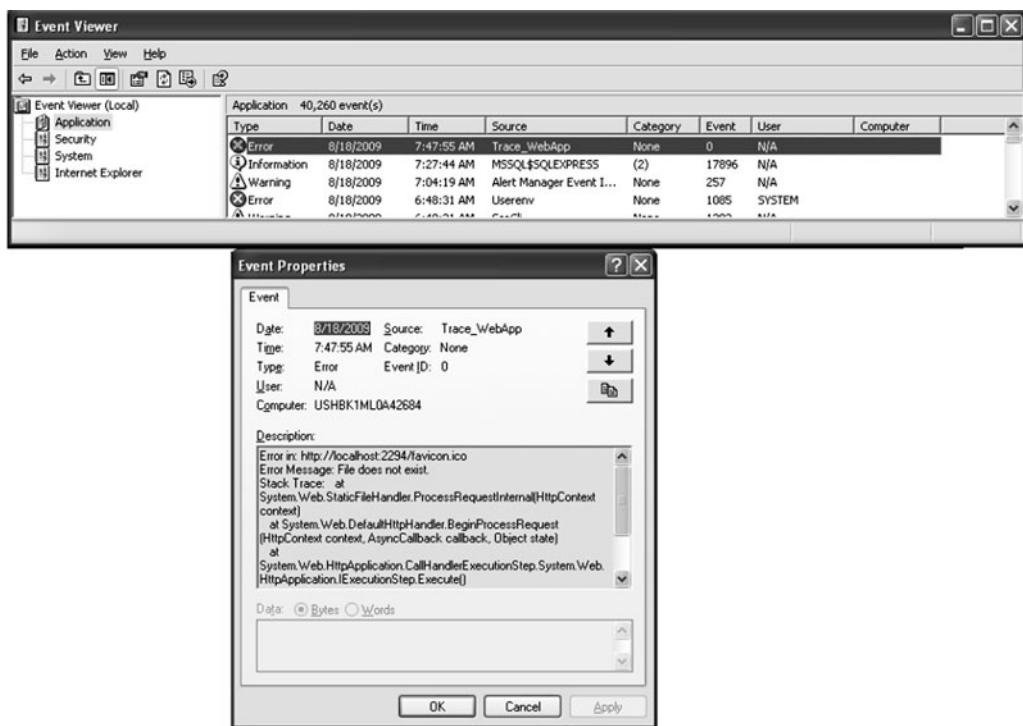
Note that this code snippet tracks error information using an event log. Remember that to use event logs, you need add a reference of the `System.Diagnostics` namespace.

5. Run the `Debug.aspx` page. The debugger will break as soon as an exception is encountered. To continue code execution and to display the error page, press F5.

To see the details of the exception in the Application event log, you need to open it using Event Viewer, as shown in Figure 12-10.

Figure 12-10

Viewing exception details



UNHANDLED EXCEPTIONS IN ASP.NET AJAX

AJAX applications contain a lot of client-side scripts, such as VB Script, J# script, or JavaScript.

You can debug client side scripts using Visual Studio or using Microsoft Script Editor.

Visual Studio provides default debugging options that enable you to debug .NET applications. Note that before starting client side script code debugging, you need to enable client-side script debugging in Internet Explorer (IE). After enabling client-side debugging in IE, you need to invoke the client-side debugger. To do so, include the keyword `debugger` in the first line of the JavaScript code. The rest of the process for debugging, watching value of local variables is similar to the method for debugging applications with server-side code.

In addition to Visual Studio, you can also use the Microsoft Script Editor (MSE) tool to debug AJAX applications. This software is freely available for download on the Internet. To use this, you just need to have IE installed on your machine.

Debugging in AJAX is somewhat challenging because you need to debug the code on the browser. For this, you can use the `sys.debug` namespace, which helps you to write trace messages. The `sys.debug` class provides methods, such as `assert` and `trace`, which you can use to trace AJAX applications.

CERTIFICATION READY?

Debug unhandled exceptions when using ASP.NET AJAX.

4.3

MORE INFORMATION

For more information on the methods of the `sys.debug` class, refer to the `Sys.Debug` class section on MSDN.

Understanding Custom and Generic Handlers

Handlers are various common classes of ASP.NET. These classes handle common requests for query strings that require a page type or service type output. For certain specific types of requests, these common classes might not be equipped to handle the requests. In such situations, ASP.NET provides you with custom handlers.

All such requests processed by ASP.NET are handled by implementation of the `IHttpHandler` interface. This handler works with various classes that handle common requests. In this section, we discuss the custom handlers supported by ASP.NET and how to implement them.

ASP.NET provides the `Page` class to process all UI operations. Because UI processing involves a lot of operations, the `Page` class has a lot of functionality built into it. ASP.NET also provides another class called the `WebService` class, which implements the details required to interpret HTTP requests as method calls. Clients can access web services provided by the application by packaging method calls in XML format. Usually, these client calls are executed with the help of the HTTP GET and POST methods.

All HTTP requests processed by ASP.NET are handled by classes that implement the `IHttpHandler` interface, which is a simple interface that consists of a method—`ProcessRequest`—and a property—`IsReusable`:

```
public void ProcessRequest(HttpContext context)
{
}

public bool IsReusable
{
    get
    {
        return false;
    }
}
```

Apart from `IHttpHandler`, ASP.NET provides several other HTTP handlers for performing common functions, such as tracing and restricting user access. These handlers can be found registered in the application `web.config` file. Handlers usually include the following items:

- File name and/or extension to which the handler applies. This is done using the `add` attribute.
- One or more verbs to which the handler applies. These verbs correspond to the HTTP specifications, such as `GET` and `POST` requests.
- The name of the .NET type assigned to handle a request.
- The `validate` attribute that specifies whether ASP.NET should load the class at startup or wait until it receives a matching request.

Here is a sample code of the `web.config` file section to register a handler:

```
<system.web>
    <httpHandlers>
        <add verb="*" path="*.aspx" validate="false"
type="Debug.CustomFormHandler,CustomFormHandlerLib">
            </add>
    </httpHandlers>
</system.web>
```

USING IHttpHandler

As already mentioned, the `IHttpHandler` interface includes the `ProcessRequest` method and the `IsReusable` property. The `IsReusable` property allows you to check whether an interface is reusable or not. The `IsReusable` should return true if the handler instance can be used multiple times. Generally, if the handler returns static content, it's probably reusable, and if the content is dynamic, it's probably not reusable. The handler contains the `ProcessRequest` method that includes a single parameter, the current `HttpContext`, which provides the context for the current request from anywhere in the application domain.

Once a request arrives at the handler through the `ProcessRequest` method, the `Trace.axd` handler responds to a `GET` request by listing the requests being tracked by the runtime. The

forbidden handler responds by blocking the forbidden source from the client. A custom web service might respond to the request by parsing the XML payload, constructing a call stack, and making a call to an internal method.

USING BUILT-IN HANDLERS

ASP.NET provides several built-in custom handlers, such as the **Trace** handler, that are implemented from **IHttpHandler**. The **Trace** handler allows you to perform tracing actions at page and application levels. When you request the trace information for an application, it is processed by the **Trace** handler, **Trace.axd**. To use the **Trace** handler, you need to turn it on in the **web.config** file by including the **trace** element.

ASP.NET caches the output of the **Trace** handler in memory, which can be viewed by browsing through the **Trace.axd** file in the virtual directory. Because this tracing functionality of ASP.NET is not a part of the standard UI processes, it is handled by a custom handler. However, IIS treats all requests as regular ASP.NET requests. Therefore, IIS will pass requests for resources with a file extension **.axd**.

USING HANDLERS AND SESSION STATE

In order to access a session from an **HttpHandler**, you need to add a marker interface. Marker interfaces have no **HttpHandler** class implementation requirements. Session states work well within the context of the **System.Web.UI.Page**. However, you need to specify a custom handler's ability to use session states by inheriting a handler from marker interfaces, which are provided by the **System.Web.SessionState** namespace. You can choose from several interface options based on the performance requirements of your page. Marker interfaces include the following:

- **IRequiresSessionState**: Specifies the HTTP handler and requires both read and write access to session state values.
- **IReadOnlySessionState**: Specifies the HTTP handler and requires only read access to session state values.

The following code example shows how **Handler1** implements **IHttpHandler** and the marker interface **IRequiresSessionState**:

```
using System.Web.SessionState;
public class Handler1: IHttpHandler, IRequiresSessionState
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "text/plain";
        context.Response.Write("Hello " + context.Session["UNAME"]);
    }
    public bool IsReusable{ get { return false;}}
}
```

Using the handler defined in this code example, you can access session states with the help of the **HttpContext** object and write a message “Hello <User Name>” using the **Response** object.

USING GENERIC HANDLERS

Like **ASPX** files, generic handlers can be compiled as just-in-time modules. These handlers have the file extension **.ashx**. Generic handlers are equivalent to custom handlers written in **C#** or Visual Basic because they contain classes that fully implement **IHttpHandler**. Since these files are compiled at runtime, to use these handlers, you just need to navigate through these files to compile them.



CREATE A GENERIC HANDLER

The following example illustrates how to create a generic handler, MyCustomHandler:

1. Open the MyCustomHandler Web site project in Visual Studio.
2. In Solution Explorer, right click the project name, and select Add New Item.
3. From the templates section, select the Generic Handler template, and name it MyCustomHandler.ashx. Visual Studio generates a handler that includes the ProcessRequest method and the IsReusable property.
4. Write the function to generate the form-handling code and call the ProcessRequest method.

The advantages of using generic handlers are that they are easy to create, convenient to use, and do not need to be configured in the web.config file or IIS. However, they have the same limitations as the ASPX and ASCX files. You need to include the handler with the whole project for it to work correctly. You can also deploy separate assemblies and share them among the enterprise as Global assemblies.



WRITE A CUSTOM HANDLER

To write a custom handler:

1. Create a new Web site from File < New > Web site, and name it MyCustomHandler. Remember to set the Location as HTTP.
2. Add a Class Library as a subproject within the solution by right clicking the Solution < Add > New Project, and selecting the Class Library. Name it LibMyCustomHandler.
3. In the solution window, select the Class1.cs file, and change the name in the Property window to MyCustomFormHandler.
4. Add a System.Web reference in the LibMyCustomHandler Class Library.
5. Add the IHttpHandler interface to the class using the following code:

MyCustomFormHandler. Adding the IHttpHandler.

6. Add HandleForm as a method having a parameter of type HttpContext. The following code shows how the HTML tags are displayed on the screen using context.Response.Write.

```
using System;
using System.Collections;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;

namespace Debug
{
    /// <summary>
    /// Summary description for $codebehindclassname$
    /// </summary>
    public class MyCustomFormHandler: IHttpHandler
    {
        public void ProcessRequest(HttpContext context)
        {
            context.Response.ContentType = "text/plain";
            ManageForm(context);
        }
    }
}
```

```
public bool IsReusable
{
    get
    {
        return false;
    }
}
public void HandleForm(HttpContext context)
{
    context.Response.Write ("<HTML><body><form>Hello
there. This is Custom Handler Response<BR> </form></body></HTML>");
}
```

7. Add the dll of the LibMyCustomHandler.dll into the reference to Web site MyCustomHandler.

8. Update the web.config file to use the MyCustomFormHandler when required.

9. Add a new section within the <system.Web> element as <httpHandler> and point the request to MyCustomFormHandler, as shown in the following code:

```
<system.Web>
    <httpHandlers>
        <add verb="*" path="*.aspx" validate="false"
type="Debug.CustomFormHandler,CustomFormHandlerLib"></add>
    </httpHandlers>
</system.Web>
```

10. Finally, create an .aspx file MyCustomHandler.aspx in the Web site.

11. Execute the Web site, and ensure that you set the MyCustomHandler.aspx as Set As Start Page. This will invoke the custom handler created to display the output, which is the custom handler response.

You can use the file with that extension to surf to the handler. Here, it is .aspx and therefore, for each file having the.aspx extension, you will get the same output, as shown in Figure 12-11.

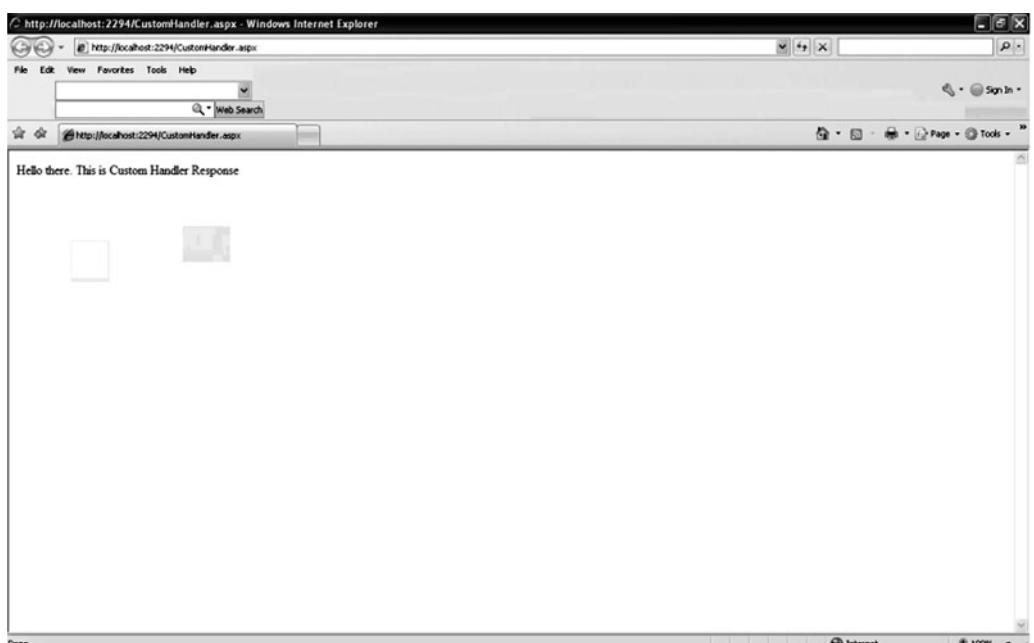
CERTIFICATION READY?

Implement the generic
handler.

7.7

Figure 12-11

Output



SKILL SUMMARY

In this lesson, you learned to troubleshoot errors. Troubleshooting an application involves tracing and monitoring. Tracing an application is the process of collecting information about web page execution, and debugging is the process of tracking an application's performance to trace and find the sources of bugs in the source code of the application. In ASP.NET, tracing can be configured at multiple levels, page level, and application level. You learned about the advantages and disadvantages of both. You learned how to turn on page tracing by enabling the `Trace` property of the `Page` class to `true`.

This lesson also provided a brief overview of Trace statements and the procedures to add and implement Trace statements in your application.

Next, you learned how to fix issues in your code by debugging. You learned about error handling, breakpoints, and error pages in this context. Finally, you learned about unhandled exceptions and using the different types of handlers for debugging. In this context, you learned about handlers, the classes that handle common requests for query strings and require a page type or service type output. ASP.NET provides the `Page` class to process all UI operations. All HTTP requests processed by ASP.NET are handled by classes that implement the `IHttpHandler` interface. It is a simple interface that consists of a method—`ProcessRequest` and a property—`IsReusable`.

ASP.NET provides several built-in custom handlers, such as the Trace handler, that are implemented from `IHttpHandler`. The Trace handler enables you to perform tracing actions at page and application levels. Generic handlers are equivalent to custom handlers written in C# or Visual Basic because they contain classes that fully implement `IHttpHandler`. The advantages of using generic handlers are that these handlers are easy to create, convenient to use, and do not need to be configured in the `web.config` file or IIS.

For the certification examination:

- Identify the types of tracing and use application and page tracing to debug Web applications.
- Debug Web applications and manage error pages.
- Manage unhandled exceptions.
- Debug deployment issues
- Explain remote debugging.
- Implement and use built-in Custom and Generic Handlers (ASHX files) and `IHttpHandler`.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ an application is the process that involves collecting information about web page execution.
2. Trace _____ configures the ASP.NET code tracing service that controls how trace results are gathered, stored, and displayed.
3. When you are executing an application, errors can occur. You can handle such errors at two levels: application level and _____ level.
4. In the _____ file, you should provide the information about the error page to be displayed.
5. The trace information for application tracing is found in a file called _____.

Matching

Match the term in Column 1 to its description in Column 2.

- a. Enabled
- b. localOnly
- c. pageOutput
- d. requestLimit

- _____ 1. Specifies whether the trace output should be restricted to the local computer.
- _____ 2. Specifies how many trace messages can be stored before removing the earlier set of trace messages.
- _____ 3. Enables or disables application-level tracing.
- _____ 4. Specifies whether the trace output should be displayed on individual pages, in addition to caching application-level messages in an individual web page.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|---|---|
| T | F 1. When your application throws an error, you need to start the tracing process. |
| T | F 2. To turn page tracing on, you need to set the <code>Trace</code> property of the <code>Page</code> class to <code>true</code> . |
| T | F 3. Application debugging can be performed either on a local computer or from a remote location. |
| T | F 4. An unhandled exception is one that occurs outside a try/catch block and is displayed at the application level. |
| T | F 5. The <code>Trace</code> handler enables you to perform tracing actions at page and application levels. |

Multiple Choice

Circle the letter or letters that correspond to the best answer or answers.

1. Which of the following options helps you track an application's performance to trace and find the sources of bugs in the source code of the application?
 - a. Tracing process
 - b. UI process
 - c. Debugging process
 - d. Personalization
2. Which of the following `trace` element attributes sets the order in which trace messages should be output to a requesting browser?
 - a. `pageOutput`
 - b. `traceMode`
 - c. `localOnly`
 - d. `requestLimit`
3. Which of the following windows displays the sequence in which the methods are called?
 - a. Call Stack window
 - b. Local window
 - c. Immediate window
 - d. Watch window
4. What is the file extension of the generic handlers?
 - a. `.ashx`
 - b. `.aspx`
 - c. `.ascx`
 - d. `.dll`

5. What are IRequiresSessionState and IReadOnlySessionState?
 - a. Generic handlers
 - b. Applications
 - c. Methods
 - d. Marker interfaces

Review Questions

1. What are unhandled exceptions?
2. What is the advantage of application tracing over page tracing?
3. What is the function of generic handlers?

■ Case Scenarios

Scenario 12-1: Using Tracing

You have developed a Web site for your customer, and there is a production date next month. You have taken care of all possible risks in the preproduction environment. You deploy your application in the production environment, and after a few weeks, a customer reports very slow performance of the Web site. How will you find the root cause of this issue?

Scenario 12-2: Using Debugging

You have done a thorough testing of various scenarios on your Web site, but after deploying it in the production environment, customers reported a bug in the Policy Issue form, which is a showstopper for the application. How will you find out the root cause of the bug?



Workplace Ready

Use HTTP Handler

You are developing an application, and your application needs to show images dynamically on the web page. These images could include some custom images or some standard images. You need to decide how it can be achieved.

Deploying and Publishing Web Applications

OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Deploying web applications.	Publish web applications.	1.5
Deploying web applications.	Debug deployment issues.	4.5
Publishing web applications.	Publish web applications.	1.5
Publishing web applications.	Monitor web applications.	4.6

KEY TERMS

web farm

Windows Installer

Deploying a web application is often the final stage of the development process. Today's technologies offer you many options to deploy your application on the web. You should choose the appropriate option to host the application based on a number of factors, such as the type of application, purpose of the application, number of users that will access it, and location and technology constraints. This lesson provides an overview of the commonly used deployment methods that you can adopt while deploying and publishing your web application.

■ Deploying Web Applications



THE BOTTOM LINE

ASP.NET supports deployment of your application on a single server or on multiple servers in a web farm. ASP.NET also enables you to support any future updates or releases of the application. You can use the options provided by ASP.NET to monitor and enhance the performance of your application through event providers and caching.

The most common method of deploying web applications is by using the Web Setup Project in Visual studio. This option allows you to package your web application to create an installer. You can distribute this installer to users using CDs or the web, and users can then run the setup file and step through a wizard to install the application.

Using Web Setup Project

The .NET Framework provides you with complete support to deploy and maintain your Web site on a single server or on a multiserver **web farm**.

In a web farm, multiple web servers store copies of the same application. This helps balance the load on each web server and allows you to increase the responsiveness of your application.

The .NET framework enables you to automatically update files between servers using any file synchronization tool. This eliminates the need for you to log on to the web server, edit registry entries, or even restart the servers to update the application.

In certain circumstances, you may need to have additional control over how your application is deployed over the web. For this, you may need to configure the web server, add registry entries, deploy files to other locations, or install prerequisites. Using a **Windows Installer** (.msi) file allows you to deploy the application with tools such as Microsoft Systems Management Server or Active Directory software distribution. The users can also download and install applications on their computer with Windows Installer. This is very useful in the Web Setup Project if the target user base for your web application is very large.

UNDERSTANDING WEB SETUP PROJECT PROPERTIES

Before learning how to use Web Setup Project, let's discuss some properties that you may need to use when deploying the application:

- **Author:** Specifies the name of the author of the application or component.
- **AddRemoveProgramIcon:** Specifies an icon to be displayed in the Add Remove programs dialog box.
- **Description:** Specifies a free form description.
- **DetectNewerInstalledVersion:** Specifies whether to check for the new version of the application during installation.
- **Manufacturer:** Specifies the name of the manufacturer of an application.
- **ManufacturerUrl:** Specifies the URL for the Web site containing the manufacturer's information.
- **ProductName:** Specifies the product name.
- **RemovePreviousVersion:** Specifies whether an installer will remove the previous version.
- **RestartWWWService:** Specifies whether IIS will be restarted when the package is installed.
- **Title:** Specifies the title for the installer.



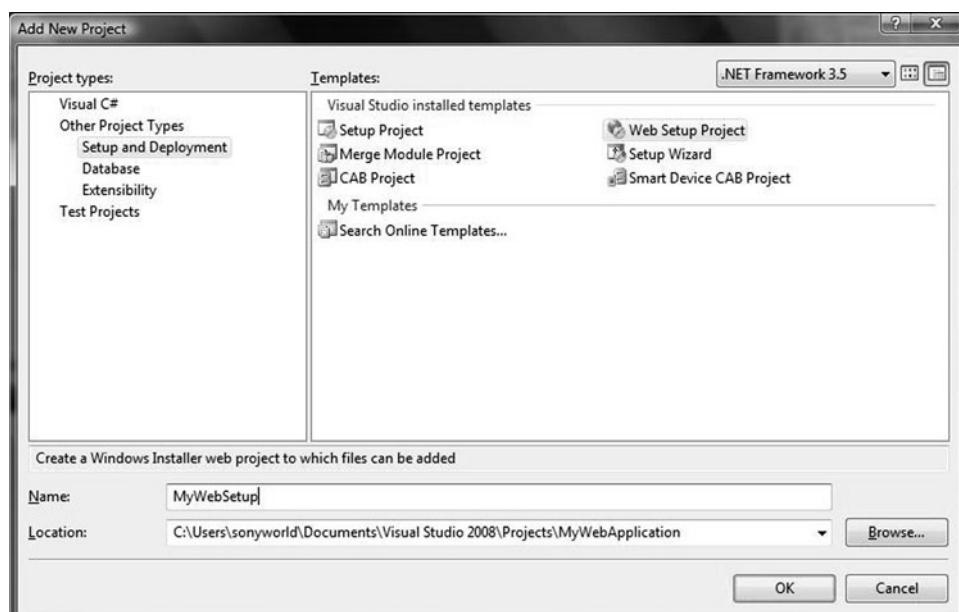
SET UP AND BUILD A WEB SETUP PROJECT

The following steps will help you to set up a Web Setup project:

1. Open the Web site in Visual Studio.
2. On the File menu, select Add > New Project.
3. Then select Project Types > Other Project Types > Setup And Deployment.
4. In Templates, choose Web Setup Project as shown in Figure 13-1.

Figure 13-1

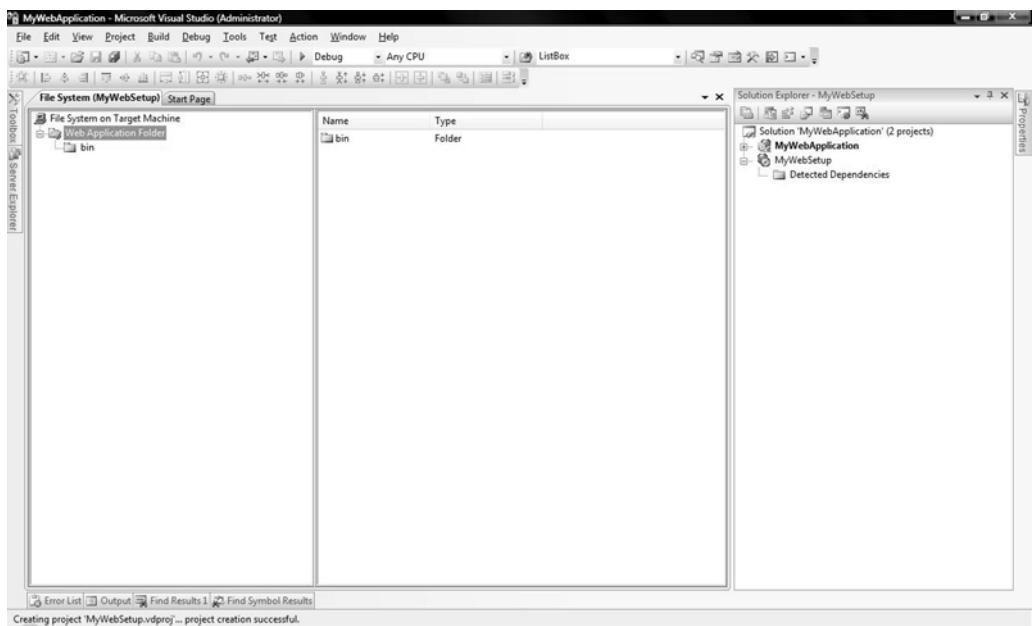
Choosing Web Setup Project



5. Type a name for your web project in the Name field and click OK. Visual Studio adds the project to your Web site and displays the File System editor.
6. Right click the Web Application Folder, and click Add > Project Output. The Add Project Output Group dialog box is displayed along with Content Files.
7. Click OK. After creating the Web Setup Project for your application, you can add additional components that you have not yet included, such as folders, files, images, and assemblies. Figure 13-2 shows the File System window for the created Web Setup Project.

Figure 13-2

File System window



To build the Web Setup Project, you should select the Web Setup Project in Solution Explorer and choose Build. Visual Studio's build output shows you where the Windows Installer file is generated. You can distribute this installer using a CD or host on the web for users.

Working with Deployment Properties and Conditions

Complex web applications with complex dependencies need custom registry entries or an administrator to configure the application. The Web Setup Project allows you to deploy web applications to meet these requirements through the following options:

- **Specifying launch conditions:** Allows you to specify the launch conditions that restrict the software/hardware configuration on which the web application can be installed. You can also check for any preinstalled components on the target computer.
- **Using custom setup:** Allows administrators to deploy and manage web applications by editing the web.config file. This enables simpler configuration at setup time by prompting users to provide specific information for custom actions.
- **Adding custom actions:** Provides flexibility and meets most setup requirements of Web Setup Project. This allows you to meet additional requirements, such as submitting registration information to a web service or validating a product key.
- **Making registry entries:** Allows you to store information on application settings. The best practice for configuring .NET Framework applications is to store settings in configuration files. However, you may need to add registry entries during setup to perform tasks such as configuring an aspect of the operating system or another application.

The .NET Framework allows you to configure deployment conditions to specify operating system versions, specific service pack levels, or other criteria. Table 13-1 lists some of these commonly used conditions.

Table 13-1

Deployment conditions

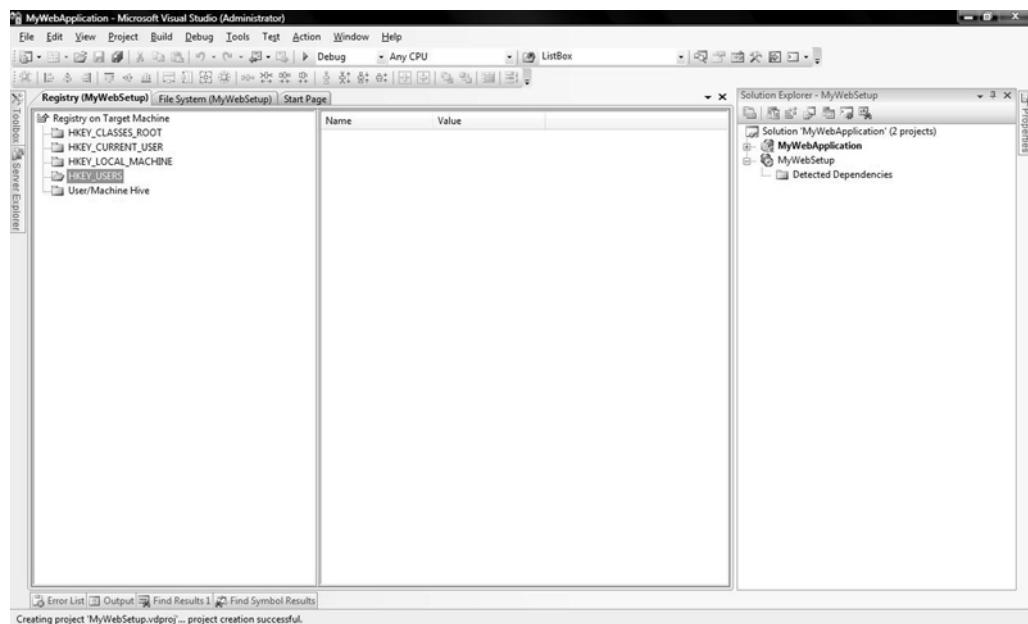
CONDITION	DESCRIPTION
VersionNT	Specifies the version number for the operating systems based on Microsoft Windows NT.
ServicePackLevel	Specifies the version number of the service pack of the operating system.
WindowsBuild	Specifies the build number of the operating system.
AdminUser	Specifies whether the user has administrative privileges.
PhysicalMemory	Specifies the size of the installed memory.

During deployment, you can also perform some other tasks, such as adding new directories to copy your application to.

You can also add new Registry entries during your installation. To do this, you need to access the registry by right clicking Web Setup Project. Then, select the Registry option from View. The Registry window is displayed as shown in Figure 13-3.

Figure 13-3

Registry view

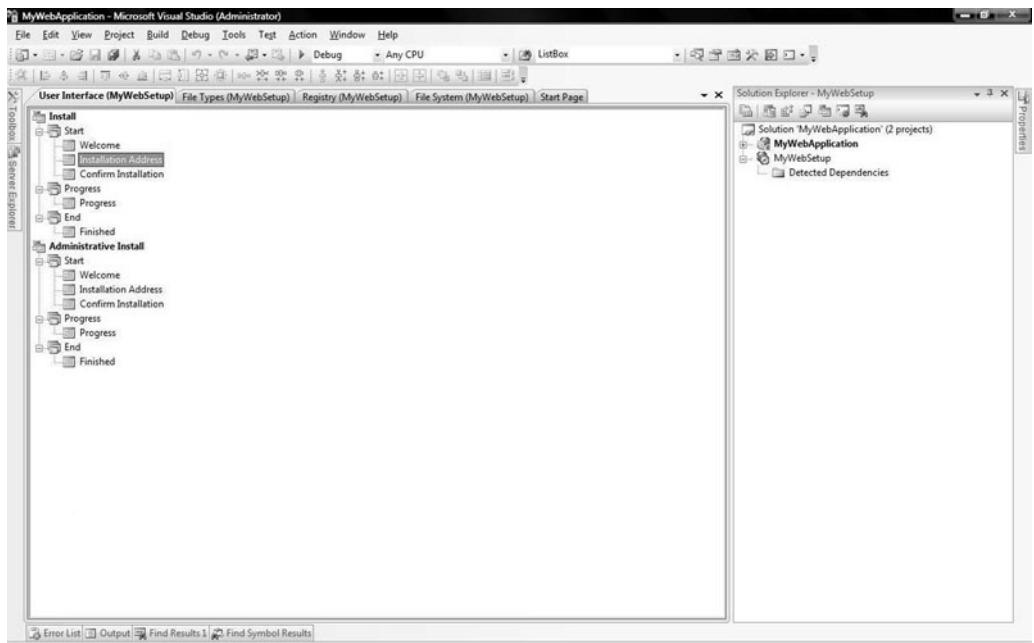


Here, you can add new registry entries or modify existing registry entries as needed.

You can also modify the user interface (UI) screens of the Web Setup Project. Select the User Interface option from the View menu by right clicking Web Setup Project. The UI screens are displayed in the existing sequence of steps as shown in Figure 13-4.

Figure 13-4

Modifying user interface

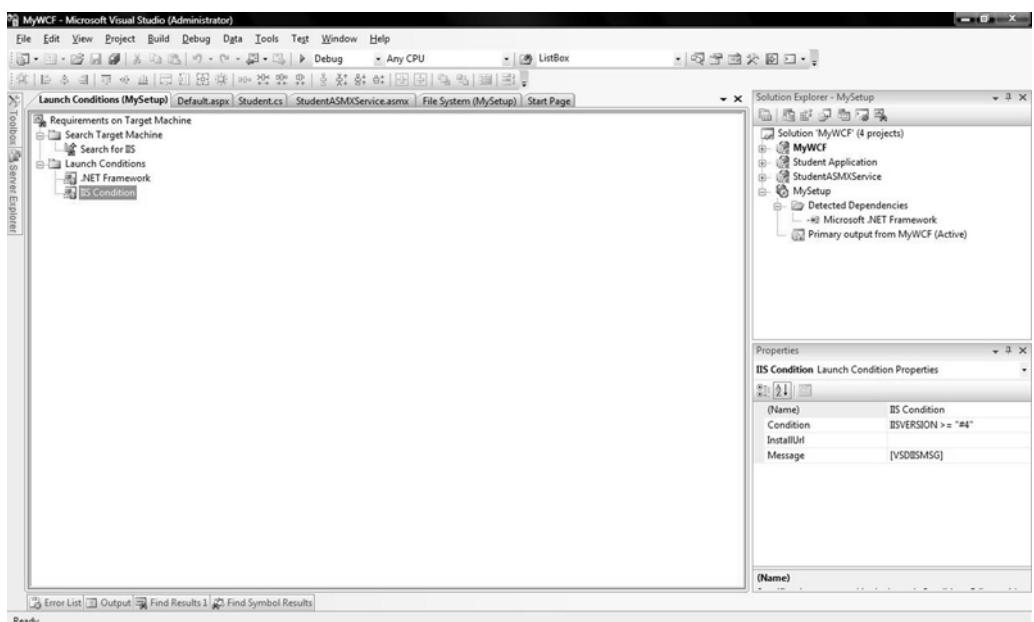


Here, you can modify the order of the screens by dragging the specific screen and dropping it to the appropriate desired place in the sequence or right click and select the Move Up or Move Down option. You can also remove screens from the predefined interface by right clicking on the screen and selecting the Delete option.

You can also specify the launch condition from the Launch Condition window. For example, you can specify the launch condition as IIS version 4 or greater as shown in Figure 13-5.

Figure 13-5

Specifying launch conditions



Here, you can check for any specific IIS version or do any kinds of system check. Based on the conditions that the Web Setup Project meets, it proceeds with the operation or aborts.

You can also use Windows Azure to develop and deploy web applications. Windows Azure is an open-platform operating system that provides development, hosting, and service management capabilities for developers. For more information on Windows Azure, refer to www.microsoft.com.

TAKE NOTE *

TAKE NOTE *

You can also use the ClickOnce technology to deploy web applications. This technology provides a simple approach to deploy and update web applications. The downside is that there can be only limited user installations. However, this is also beneficial in enhancing the security of web applications. To know more about ClickOnce deployment, refer to MSDN.

CERTIFICATION READY?

Publish web

applications.

1.5



To know more about deployment issues, refer to Lesson 12, “Troubleshooting and Debugging Web Applications.”

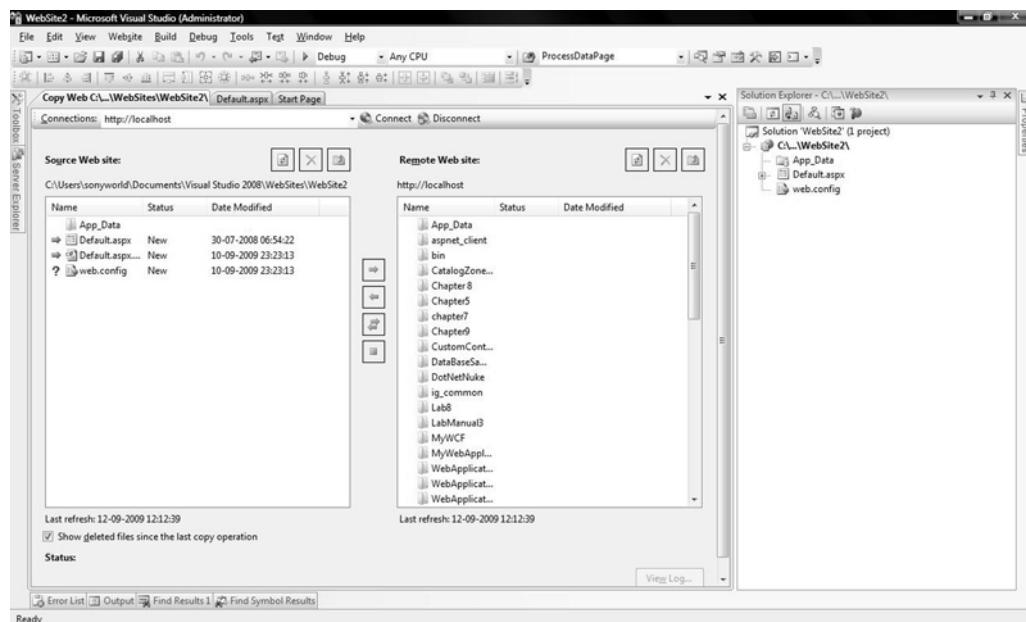
Using the Copy Web Tool

Using a web project for updates to web applications that are targeted at a smaller number of users with a more homogenous configuration of computers may not be appropriate. A simpler way to perform the updates is to edit the web application directly on the web server. These changes are immediately implemented in your production web application.

For example, consider that you want to use some content from an already published web application in your application. You can use the Copy Web tool to copy the appropriate content in your application. However, the published application may have some existing bugs, which would then get transferred to your application. The unintentional introduction bugs into your application is the primary drawback of this tool. To avoid this risk, you can edit a local copy of the web application, and if it meets the quality requirements, you can publish the changes to the production web server using the Copy Web tool. Figure 13-6 shows the Copy Web tool.

Figure 13-6

The Copy Web tool



The Copy Web tool allows you to publish changes to an application. You can publish these changes from a test server to a production web server, or between any two web servers. You can copy individual files or an entire Web site to or from a source Web site to a remote Web site. The Copy Web tool provides you with the options to synchronize files and detect versioning conflicts in those files. However, you cannot use the Copy Web tool to merge changes in two files.

When there is a conflict in the versions of the files you are copying, you need to ensure that you are copying the correct version. This is because Visual Studio does not have the capability to merge two files. Therefore, you need to analyze the conflicting files to determine which file you need to keep, or incorporate the changes manually into one of the files, and then copy that file.

When configuring the Copy Web tool to work with a remote Web site, you will be provided with the following options:

- **File system:** This is the destination Web site on a local hard drive or a shared folder to which the web application files will be copied.
- **Local IIS:** This is the destination web application that is running within IIS on your local computer. This interface also provides you with an option to create a new web application on your local server.
- **FTP site:** This is the destination on the remote Web site where the web server is configured to run a File Transfer Protocol (FTP) server. This allows uploads and downloads to the web application folder.
- **Remote site:** This is an interface where you can transfer files to and from a remote web application using FrontPage extensions, if you have already configured the web server to allow this type of update. You can use this interface to create a new web application on your server.

After you connect to the web server, you can copy or synchronize files between the source and the remote Web site in the following ways:

- Select files in either the source or remote location, and click the directional Copy Selected Files button.
- To copy the entire Web site, right click the Source Web Site pane, and then click Copy Site To Remote. Similarly, to copy the remote Web site to the source Web site, right click the Remote Web Site pane, and then click Copy Site To Source.
- To synchronize individual files, select the files in either the source or remote Web site, and then click Synchronize Selected Files.
- To synchronize the entire site, right click the Source Web Site pane, and then click Synchronize Site.

Using Web Applications Precompilation

ASP.NET offers you two ways to compile your web application: web application project and Web site projects.

In Web site projects, you need to copy the source file along with the related resources on the web server. These files are compiled on demand at the server. This leads to an increase in response time required to present the requested page to the client on first load as well as a potential risk of intellectual property caused by availability of source code on the server.

Web application projects overcome these limitations by requiring you to precompile the code before deployment. Precompiling a web application generates assemblies, configuration information, and other resources required for the web application. Using precompilation, you can see the errors and fix them before deploying. In this manner, you can avoid making your clients wait when accessing the error-free application. In addition, you do not need to copy the source code on the server, and this reduces the risk of compromising intellectual property.

In a precompiled web application, you need to compile the entire application in a single assembly before deploying it. During deployment, the precompiled assembly and its dependent resources are copied to the server. When a request for a page is received, ASP.NET creates an instance of that page's code-behind class that generates and sends the page to the client. This enables you to protect the source code of your application.

UNDERSTANDING PRECOMPILATION OPTIONS

ASP.NET provides several ways for you to precompile a web application:

- **Performance precompilation:** Allows you to compile the ASP.NET file types in place within an application. Other file types, such as HTML and images, are not compiled. The ASP.NET compiler creates assemblies for in-place precompiled files and stores them in a special folder. Then, ASP.NET uses the assemblies from this folder to fulfill

any request from a web page. This is very useful when only certain sections of your web application need to be updated frequently. The previously compiled pages that have not changed are not compiled again. This vastly reduces the compile time and enables you to compile the web application as often as required. For this, you need to have source code on the target server.

- **Deployment precompilation:** Allows you to create an executable version of your web application that contains assemblies, configuration information, and resources. Precompiling an application for deployment allows you to ensure maximum security because it does not require the source code on the target computer to be copied; only static files such as HTML pages and images are copied. The web applications precompiled for deployment are not easy to update. If you modify any file, then you need to recompile the entire site. If you add any new files to a precompiled web application, then you need to compile the entire application from scratch before deploying it.

Publishing Precompiled Web Applications

ASP.NET provides extensive support to precompile an application in place as well as for deployment.



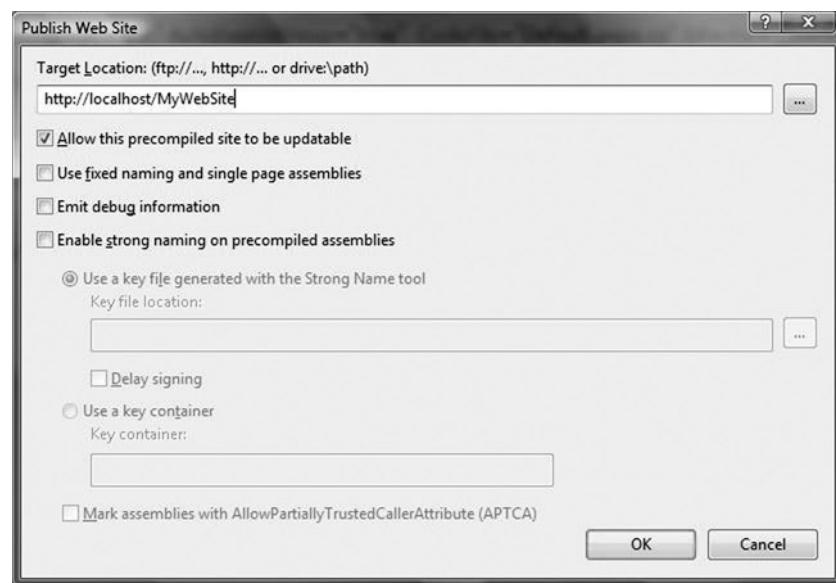
PRECOMPILE AN APPLICATION

To precompile an application:

1. In Visual Studio, open the MyWebsite web application.
2. In Solution Explorer, right click on your application, and select Publish. The Publish Web dialog box is displayed as shown in Figure 13-7.

Figure 13-7

Published Web dialog box



3. Specify a location to publish your Web site. For example, you can select from FTP, HTTP, or File system project. The path for HTTP has been specified.
4. Specify any one of the following options:
 - **Allow This Precompiled Web Site To Be Updatable:** Select this option if you want to update the Web site later.
 - **Use Fixed Naming and Single Page Assemblies:** Select this option to specify fixed naming.

X REF

To know more about precompilation, refer to Lesson 12, “Troubleshooting and Debugging Web Applications.”

- **Emit Debug information:** Select this option to generate the debugging information file (.pdb file) in the target output file after the Web site is published.
 - **Enable Strong Naming on Precompiled Assemblies:** Select this option if you wish to implement more security and avoid renaming registries or namespaces.
5. Click the OK button to compile and publish your Web site. The Web site will be published as an HTTP site at the target location.

MORE INFORMATION

For more information on Windows Installer deployment, refer to the ASP.NET Precompilation Overview section on MSDN.

CERTIFICATION READY?
Debug deployment
issues.
4.5

■ Publishing Web Applications

THE BOTTOM LINE

ASP.NET offers several options for you to publish, monitor, and enhance the performance of your applications through event providers and caching.

Visual Studio provides you with several options to build a web application. These options define how the files are stored on the web server, including file structure and layout of the Web site. These options include HTTP project, FTP project, and File system. You can specify the path where you wish to publish in the Target location field of the Publish Web dialog box.

Using File System Project

Visual Studio allows you to store and test the project files in a folder on your local hard drive or a network drive. Instead of creating an IIS application, you can use the file system-based project to develop and test your application. This type of project runs locally without IIS. Therefore, if you do not have IIS access, then you can use this type of project easily. The layout for a file system-based project is similar to an ASP.NET project, where the web pages appear in the root folder or any subfolder of the application. Using this project type you can develop and test the web page on your computer before deploying it on the Web site. Note that the final deployment for production use should be on IIS. You can deploy these files using the Copy Web or the Publish Web tools in Visual Studio.

Using HTTP Project

Visual Studio allows you to test your web application by hosting it on a web server before moving it to the production environment. The HTTP project allows you to test the complete web application just like how it would work on production, including resolution of file paths. To use the HTTP project, you need to have IIS running on your computer and the necessary permissions to update the files.

Visual Studio creates a virtual directory on the HTTP server and uses the web server to intercept requests during development. This virtual directory should point to the location where you have the source code on your computer. Using the HTTP server enables you to test web applications that use server-side services, such as ISAPI filters and application pooling. Visual Studio allows you to copy the web application files to the HTTP server using the Copy Web or Publish Web tool.

Using FTP Project

Visual Studio allows you to use FTP service to quickly access web project files that are stored on another web server. FTP servers are physical machines that are separate from the web server, and they do not host web sites. They only allow you to copy files that make up a Web site to and from the server. To use this FTP service, the FTP server should be activated and you should have necessary permissions to read/write on that server. This is independent of the HTTP server, which displays the web page for the browser. Both servers, FTP and HTTP, can reside on the same computer.

CERTIFICATION READY?

Publish web applications.
4.5

Using FTP service, you can create and edit files just like a local file. When you access the files from an FTP server, a copy of that file is copied to your local computer. When you save the file after editing it, the modified file is then copied back to the FTP server. With FTP, there is also the possibility of changes being overwritten when multiple developers access the source files.

Monitoring Application Health

ASP.NET provides health-monitoring features that you can use to monitor an application so that you are aware of any security issues or failures. You can use the classes provided by the `System.Web.Management` namespace to do so.

To monitor the application, you need to create events and then monitor or listen to those events. You use event viewers to listen to those events. The Web event classes, such as `WebBaseEvent`, `WebErrorEvent`, and `WebFailureAuditEvent` will help you to create events. ASP.NET also provides listeners that you can use to collect information about the events. Some examples of listeners provided by ASP.NET are `SqlWebEventProvider` and `EventLogWebEventsProvider`. Once you have created events and event listeners, you need to configure health monitoring in the `web.config` file. To do this, set the `enabled` attribute of the `<healthMonitoring>` element to `true`. You can also add rules to the `web.config` file to start the health-monitoring process.

CONFIGURING WEB APPLICATION HEALTH MONITORING

You can enable the web application health monitoring by configuring the `<healthMonitoring>` section in the `system.web` section of the `web.config` file. You should set the `enabled` attribute to `true`.

Configure Web Event and Provider

You can configure providers when you enable health monitoring. The providers are added by default. To configure a custom web event and provider:

1. Add the `healthMonitoring` section under the `system.web` section in the `web.config` file.
2. Add the `eventMappings` element to the `healthMonitoring` element.
3. Modify the `providers` element under the `healthMonitoring` tag, if required, to specify the health monitoring listeners and specific providers.
4. Add the `rule` element in the `healthMonitoring` tag. Rules are used to indicate associations between a web event and listeners.

The following configuration shows you how to use `EventLogProviders` to turn on heartbeat events:

```
<healthMonitoring enabled="true" heartbeatInterval="2">
  <rules>
    <add name="HeartBeat" eventName="Heartbeats"
        provider="EventLogProvider" profile="Default" minInstances="1"
        maxLimit="Infinite" minInterval="00:30:00"/>
  </rules>
</healthMonitoring>
```

MORE INFORMATION

For more information on health monitoring, refer to the ASP.NET Health Monitoring Overview section on MSDN.

In this configuration, the `heartbeatInterval` specifies the time interval in seconds for how often the `WebHeartbeatEvent` is raised. The `provider` element specifies the provider that processes the events. The `rules` element maps the event to the specified providers. The `minInterval` specifies the time span between two events, while `maxLimit` defines the maximum number of the rule instances that generates the provider notification, and `minInstances` identifies the minimum number of rule instances within a given application before the event notification is triggered.

Note that you can also make your own custom configuration for providers. You would typically configure custom providers if you wish to use additional providers than the one mentioned in the `web.config` file. You need to add the providers in the `<providers>` tag.

**DEPLOY THE WEB SETUP PROJECT**

Consider that you've created a web application for a consultancy, such as a personnel placement firm. This application allows job seekers to search for the specific job and allows them to upload their resume and apply for the specific job. The application also allows recruiters to search for specific skilled resources per their requirements. You've already completed the development of this web application. You need to create an installer that will install this product and remove the previous version. To do this:

1. Create your web application.
2. Right click the solution, and select Add New Project.
3. From the Add New Project window, expand Other Project Types, and select the Setup and Deployment option.
4. From the list of templates displayed, select the Web Setup Project template, and name it `MySetup`.
5. In the File System window Web Application Folder, right click and select Add Project Output.
6. Select your web application project in the Add Project drop-down list.
7. Then select Primary Output, and click OK.
8. Select the `MySetup` project, open the Properties window, and specify the product name `MyConsultancy`.
9. Set the Manufacturer to `MyCompany`.
10. Set the RemovePreviousVersion to True.
11. Save and rebuild the Web Setup Project.

Now, the setup is created. When the user runs this setup, the wizard opens, as shown in Figure 13-8.

CERTIFICATION READY?
Monitor web applications.
4.6

Figure 13-8

Web Setup Project template



SKILL SUMMARY

In this lesson, you learned that you can use Web Setup Project provided by Visual Studio to deploy your Web site. This lesson discussed how to set up and build a web project. The lesson also covered the different properties and conditions that you can set during deployment. Next, you learned about using the Copy Web tool to copy files to other locations. You learned about the publishing models in this regard. Visual Studio provides several models for publishing, such as HTTP sites that use IIS, file systems that exist in the local file system, and FTP sites.

Finally, this lesson provided an overview of precompilation and the procedure to publish precompiled applications. You learned how to monitor the health of web applications in this context.

For the certification exam:

- Use web setup project and the copy web tool to deploy web applications.
- Use Visual Studio to precompile and debug web applications during deployment.
- Use Visual Studio to publish web applications.
- Use Visual Studio to monitor the health of web applications.

■ Knowledge Assessment

Fill in the Blank

Complete the following sentences by writing the correct word or words in the blanks provided.

1. _____ precompilation allows you to compile the ASP.NET files types in place within an application.
2. The _____ tool provides you with the options to detect versioning conflicts in the files.
3. A _____ server only allows you to copy files that make up a Web site to and from the server.
4. Making _____ entries allows you to store information on application settings.

True / False

Circle T if the statement is true or F if the statement is false.

- | | |
|----------|---|
| T | F 1. You can use the Copy Web tool to merge changes in two files. |
| T | F 2. You can configure deployment conditions to specify operating system versions. |
| T | F 3. Using the Copy Web tool, you can create an installer. |
| T | F 4. During performance precompilation, HTML files and images are precompiled. |
| T | F 5. FTP servers help in hosting Web sites. |

Multiple Choice

Circle the letter that corresponds to the best answer.

1. Which of the following options allows you to store information on application settings?
 - a. Making Registry Entries
 - b. Adding Custom Actions
 - c. Using Custom Setup
 - d. Specifying Launch Conditions

2. Using which of the following interfaces, can you create a new web application on your server?
 - a. File System
 - b. Remote Site
 - c. Local IIS
 - d. FTP Site
3. Which of the following servers enables you to test web applications that use server-side services?
 - a. FTP Project
 - b. File System Project
 - c. Web Services Project
 - d. HTTP Project
4. Which of the following precompilations allows you to create an executable version of your web application that contains assemblies and resources?
 - a. Executable
 - b. Deployment
 - c. Performance
 - d. Custom

Review Questions

1. How would you balance loads among multiple web servers in a web farm?
2. You can modify a web application by editing it directly on the web server. How would you avoid including bugs in your application in this scenario?

■ Case Scenario

Scenario 13-1: Using the Copy Web Tool

You are developing a Web site for your office that allows a limited number of users to access it and book cars when leaving the office. How will you deploy the Web site considering that FTP support is not available?



Workplace Ready

Selecting the Right Deployment Method

You are developing an IIS-hosted web application for a health management system. You want to sell it to your customers through an existing Web site. The software can be purchased in two ways. A new customer may host the application in the existing environment, while existing customers may upgrade to the already hosted site. How will you decide on the tool to deploy the application? Hint: You may not need to host an entire Web site.

Object-Oriented Programming

Human beings use language for communication. There are many languages, and each is unique in its own way. Just like humans, computers are also designed to understand languages. These languages are called programming languages.

■ Overview of Programming Languages



A programming language translates a specific set of instructions into a program that a computer can understand. The computer then runs the program to follow the instructions built into the program.

Each programming language uses its own rules and structure to write a program. Each programming language also has its own traits that define the behavior of the programming language. In general, programming languages have the following traits:

- **Function:** The main function of a programming language is to create computer programs that are intended to provide a result.
- **Behavior:** A programming language is defined by its behavior to perform a function and can be used by humans to communicate with a computer or to communicate between devices. The behavior of the programming language defines an application's behavior. A programming language can be used to develop a game, which will have certain behavioral attributes. For example, in a game, a timer may complete a specific stage if the player has met specific conditions.
- **Syntax:** This element is used to define a set of instructions. Most programming languages define syntax in the form of text, which is easily usable and understandable by humans.

Programming languages are unique based on the programming paradigm they represent because each programming language typically represents only one type of programming paradigm. The programming paradigm defines how the code will be executed; however, the paradigm is not a limitation for all programming languages. For example, Java is a programming language that represents two different programming paradigms: procedural and object oriented.

The three most widely accepted programming paradigms are:

- **Procedural:** Contains code in sections, which can be defined as functions. The code set is written like a big program. There are multiple functions that are part of the code set. Each function has a starting and ending point, which flows in a sequence. For the complete code set to provide results, each function must provide results. Each function must finish for the next function to start. FORTRAN and Basic are two programming languages that use procedural paradigm. Structured programming paradigm is a subset of procedural programming that uses smaller functions to form a larger function. You

use global variables that can be shared across multiple functions to perform a specific task. C and Pascal are two examples of languages where structured programming paradigms are used.

- **Object oriented:** Uses objects. Unlike the other paradigms, you base everything on the objects, rather than using actions. An object is defined by a data field structure and the methods that are applied on the data structure. In this paradigm, multiple objects collectively form a program. An example of object-oriented language is C++. An object can be created from a class, which serves as the template for the object. You can create multiple objects from a single class. Specifications of an object can be defined by a method, which can be private, protected, or public. Private methods are limited to a specific class from which the object was based. Public methods are accessible by other classes. Protected methods are limited to the current assembly and are accessible to the derived classes only. You can also obtain specification of an object from the object property.
- **Functional:** A set of ideas, rather than a set of guidelines, that you need to follow. It is similar to object-oriented programming. However, in this paradigm, functions are used for any computation and for breaking the code into smaller and reusable chunks instead of objects.

Elements of OOP

Object-oriented programming (OOP) is a concept on which many programming languages are based. It is now by far the most accepted programming paradigm used across multiple programming languages. In OOP, as in the procedural paradigm, the programming code acts on data rather than on the logic. Every aspect of OOP revolves around objects, which are defined by the use of classes.

OOP is made up of multiple elements. Let's learn about these elements.

CLASSES

A class acts as a template or blueprint that defines common characteristics among its objects. If you need to create multiple objects of a similar kind, then you use a class. For example, a mobile phone would be a class. This class will have certain characteristics that would be common across all mobile phones. For example, all mobile phones will have the ability to make or receive a call.

A class serves as the base for objects. The class also defines the description of objects that are created from it. Once a class is defined, you can create multiple objects from the class. This saves you the effort of recreating the class every time an object needs to be created. In addition, if you create a new type every time an object is created, these instances will not be of the same type. For example, when a mobile phone with specific characteristics is designed, you can create multiple mobile phones from the same design, without having to recreate the entire design over again. In this example, the `MobilePhone` class can have `GSM` and `CDMA` as subclasses. A subclass is a class that is defined in relation to another class. There are certain behaviors and states that are inherited by the subclass from the parent class. For instance, each type of mobile phone in `GSM` or `CDMA` subclass will have a receive and disconnect button.

The following code shows a simple class in C# named `MobilePhone`. This class has two constructors: a default constructor and a parameterized constructor. The `MobilePhone` class also has a private variable `mobileType`, which you will set in the constructor:

```
public class MobilePhone
{
    // Variable
    private string mobileType;
    // Default Constructor
    public MobilePhone()
    {
        MobileType = "CDMA";
    }
}
```

```

// Constructor
public MobilePhone(string type)
{
    this.MobileType = type;
}

// Method
public bool MakeACall()
{
    return true;
}

```

OBJECT

Objects are the core element of OOP. Every object is independent and has the capability of acting on its own without relying on any other object. An object is part of a class. Each object from the same class carries a specific set of characteristics. In an object-oriented program, each object is assigned a responsibility to perform a task and can also perform a task on behalf of another object when the responsibility for that task has been delegated to it. Each object can be defined with its state and behavior:

- **State:** Defines the attributes of an object. For example, the mobile phone object will have color, size, and shape as the state of the object. A state is also known as a property.
- **Behavior:** Defines the tasks that can be performed by the object. The ability to dial a call, receive a call, and send messages constitutes the behavior of the mobile phone object. A behavior is also known as method.

Extending the mobile phone example, you can assume a specific brand of mobile phone to be an object. This object would have certain common features, such as the ability to use a specific color set or a specific type of button.

For example:

```

class Phone
{
    static void Main(string[] args)
    {
        // Creating an object of MobilePhone
        MobilePhone mobilePhone = new MobilePhone();
    }
}

```

In this code, **Phone** is the class while **mobilePhone** is the object of the **MobilePhone** class.

METHODS

Methods, which are also known as functions, contain data declarations and a sequence of instructions. Data declarations are local to methods and cannot be scoped outside of them. Each object has a set of abilities, which help the object perform a specific task. For instance, a GSM mobile phone has the ability to make a phone call. The GSM mobile phone can have many other abilities, such as the ability to receive a phone call and store a message. In the code provided earlier, **Main()** is the method.

VARIABLES

Along with methods or functions, variables are also part of objects. You can define one or more variables for an object. The variable defines a piece of information that is declared for the object. For instance, a mobile phone's supported frequency can be defined as a variable.

OOPS Concept

OOP is based on multiple concepts that bind the objects and classes together. Let's look at these concepts.

OOP is based on the following concepts:

- Inheritance
- Encapsulation
- Abstraction
- Polymorphism

INHERITANCE

In inheritance, an object acquires the properties of the object that is immediately higher in the hierarchy. Inheritance reduces the effort required to redesign the properties of each object that shares a similar nature. In this example, all Nokia phones share the characteristics provided by GSM mobile phones.

A subclass inherits most of its properties from a class, which can be considered a parent class. A parent class is also known as a superclass. You can also define the subclass to have some unique attributes that it can use to override certain attributes it inherited from the superclass.

C# supports two types of inheritance:

- Implementation inheritance
- Interface inheritance

C# does not support multiple inheritance. A class cannot be derived from more than one class. However, a class can be implemented from multiple interfaces.

You can define inheritance using the following syntax:

```
class MobilePhone
{
    public virtual void MakeACall()
    {
    }
    public virtual void ReceiveACall()
    {
    }
}

class GSM: MobilePhone
{
    public override void MakeACall()
    {
        base.MakeACall();
    }
    public override void ReceiveACall()
    {
        base.ReceiveACall();
    }
}

class CDMA: MobilePhone
{
    public override void MakeACall()
    {
        base.MakeACall();
    }
}
```

This syntax specifies that the `GSM` and `CDMA` classes are derived from the `MobilePhone` class. Here is another syntax that defines inheritance:

```
interface interface1
{
    void CallOnHold();
}
interface interface2
{
    void CallDisconnect();
}
class MobilePhone
{
    public void MakeACall()
    {
    }
    public void ReceiveACall(bool val)
    {
    }
}
class GSM: MobilePhone,interface1,interface2
{
    #region interface1 Members
    public void CallOnHold()
    {
        throw new NotImplementedException();
    }
    #endregion
    #region interface2 Members
    public void CallDisconnect()
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

The above syntax specifies that the `GSM` class has been derived from `Interface1` and `Interface2`.

ENCAPSULATION

Encapsulation means hiding complex and nonessential details. You can use public methods, private methods, and private instance variables. Using encapsulation, you can bind your code with the data. You use variables in a function that relates to an object. When you protect the variables being used, you encapsulate them, which acts like a wrapper around the variables defined for an object. The wrapper is a function or a method. When you apply the wrapper, the other objects in the class cannot access the variables.

For instance, consider the example of a mobile device. Each component is meant to perform a specific function. You can only receive calls with the receive button. The volume control button will only allow you to increase or decrease volume; you cannot use this button to modify the display resolution of the mobile phone. The display resolution is handled by another button, which invokes a specific function to call certain variables:

```
class Phone
{
    static void Main(string[] args)
    {
        MobilePhone mobilePhone = new MobilePhone();
```

```
        mobilePhone.MakeACall(true);
        Console.WriteLine("Call status is:" + mobilePhone.MakeACall().
ToString());
    }
}
class MobilePhone
{
    private bool callStatus;
    public bool MakeACall()
    {
        return callStatus;
    }
    public void ReceiveACall(bool val)
    {
        callStatus = val;
    }
}
```

In the previous code sample, you have the `MobilePhone` class that has `callStatus` as a private variable and `ReceiveACall` and `MakeACall` as public methods. The code prevents the `callStatus` method from being exposed to the external world by setting it as private. When you create an object of the `MobilePhone` class, you will find the `MakeACall` and `ReceiveACall` methods exposed in the `Main` method and not `callStatus`.

You can also achieve this by having `Get` and `Set` properties for `callStatus`:

```
class MobilePhone
{
    private bool callStatus;
    public bool CallStatus
    {
        get { return callStatus; }
        set { callStatus = value; }
    }
}
```

ABSTRACTION

Abstraction is an idea or a concept used to represent only the essential feature of an object as a single unit. For example, a person operating a computer may not know the internal functionality of the computer, but he or she should know how to operate the mouse and the keyboard.

Another example is a GSM mobile phone that has multiple pieces, such as a keyboard, display, and buttons. Although these are visible, they are not important to the user. As a result, such details are ignored, and only a GSM mobile phone set is considered as an object.

In OOP, abstraction is implemented as a class, which basically defines a set of objects with their own states and behaviors. Each class can also have subclasses. The class that has subclasses is also called the superclass, and it defines the state of the subclasses.

You can apply abstraction in multiple ways—using functionality types, appearance, or structure. The use of abstraction makes it easy to manage multiple complex pieces. You combine multiple complex pieces to build a single piece that is easily manageable. The complex pieces can further be divided into many more complex pieces. The complex pieces can be represented in hierarchical form.

The following code depicts abstraction:

```
public abstract class MobilePhone
{
    public abstract void MakeACall();
```

```

        public abstract void ReceiveACall();
    }

    class GSM: MobilePhone
    {
        public override void ReceiveACall()
        {
            throw new NotImplementedException();
        }
        public override void MakeACall()
        {
            throw new NotImplementedException();
        }
    }
}

```

In this code sample, you have defined `MobilePhone` as an abstract class by mentioning the `abstract` keyword while defining the class. You have also defined abstract methods, `MakeACall` and `ReceiveACall`. The `MobilePhone` abstract class is derived by the `GSM` class and abstract methods are implemented in the derived class.

POLYMORPHISM

Polymorphism allows an object to have multiple types of behavior. Objects in different classes can respond to methods that have similar names. For instance, a call button on a mobile phone can be used to receive a call or to provide a list of previously called numbers. Depending on different behaviors, different results are provided. You use polymorphism to redefine the methods or functions. You can use the following in OOP to perform polymorphism:

- **Method overloading:** Defining methods with the same name.
- **Operator overloading:** Using operators to behave differently based on the arguments they take.
- **Method overriding:** Allowing a subclass to override a method provided by a superclass.

Overloading is when you use a method, operator, or function with the same name but use different parameters to perform different functions. Overriding is modifying or replacing the implementation of the parent class with a new class. Parent classes with virtual or abstract members allow derived classes to override them, as shown in the following code:

```

public class MobilePhone
{
    private bool callStatus;
    public bool CallStatus
    {
        get { return callStatus; }
        set { callStatus = value; }
    }
    public virtual bool MakeACall()
    {
        return false;
    }
}

class CDMA: MobilePhone
{
    public override bool MakeACall()
    {
        return true;
    }
}

```

In this code sample, you have defined `MobilePhone` as an abstract class and the `MakeACall` method as virtual so that you can override them in the derived class. The `CDMA` class is derived from `MobilePhone`, and you can override the `MakeACall` method to always return a `true` value.

OOP Languages

To be able to use OOP, you need to use OOP language (OOPL).

OOPL uses the concept of OOP to successfully execute code. OOPL has the following characteristics:

- It focuses more on the data and uses the objects concept. There is no focus on procedures.
- Each program is modularized into specific pieces. These pieces are known as objects, which can be reusable as well.
- Objects are characterized by data structures, which collectively bind the functions operating on an object.
- Objects can be defined so that they are able to communicate with each other. Communication occurs through the use of methods.
- Modification of objects, which may include adding or removing objects, is possible.

The most popular OOPLs are C++, C#, Visual Basic, and Java. OOP is also being used in scripting languages, such as Ruby and Python.

TAKE NOTE *

The programming language Simula 67 first introduced objects in its programming structure. However, SmallTalk can be considered the first programming language based on object orientation.

SKILL SUMMARY

Object-oriented programming is a paradigm that uses objects, classes, and methods to develop applications. Object orientation is based on encapsulation, polymorphism, abstraction, and inheritance.

Appendix B

Microsoft .NET Framework 3.5, ASP.NET Application Development

OBJECTIVE DOMAIN	SKILL NUMBER	LESSON NUMBER
Configuring and Deploying Web Applications		
Configure providers.	1.1	7, 9
Configure authentication, authorization, and impersonation.	1.2	9
Configure projects, solutions, and reference assemblies.	1.3	1
Configure session state by using Microsoft SQL Server, State Server, or InProc.	1.4	7
Publish web applications.	1.5	13
Configure application pools.	1.6	1
Compile an application by using Visual Studio or command-line tools.	1.7	1
Consuming and Creating Server Controls		
Implement data-bound controls.	2.1	2, 7
Load user controls dynamically.	2.2	3
Create and consume custom controls.	2.3	3
Implement client-side validation and server-side validation.	2.4	7
Consume standard controls.	2.5	2
Working with Data and Services		
Read and write XML data.	3.1	5, 10
Manipulate data by using DataSet and DataReader objects.	3.2	4
Call a Windows Communication Foundation (WCF) service or a web service from an ASP.NET web page.	3.3	10
Implement a DataSource control.	3.4	5
Bind controls to data by using data-binding syntax.	3.5	2
Troubleshooting and Debugging Web Applications		
Configure debugging and custom errors.	4.1	12
Set up an environment to perform remote debugging.	4.2	12
Debug unhandled exceptions when using ASP.NET AJAX.	4.3	12
Implement tracing of a web application.	4.4	12

OBJECTIVE DOMAIN	SKILL NUMBER	LESSON NUMBER
Debug deployment issues.	4.5	12, 13
Monitor web applications.	4.6	13
Working with ASP.NET AJAX and Client-Side Scripting		
Implement web forms by using ASP.NET AJAX.	5.1	6
Interact with the ASP.NET AJAX client-side library.	5.2	6
Consume services from client scripts.	5.3	6, 10
Create and register client script.	5.4	6
Targeting Mobile Devices		
Access device capabilities.	6.1	11
Control device-specific rendering.	6.2	11
Add mobile web controls to a page.	6.3	11
Implement control adapters.	6.4	11
Programming Web Applications		
Customize the layout and appearance of a web page.	7.1	1, 3
Work with ASP.NET intrinsic objects.	7.2	1
Implement globalization and accessibility.	7.3	8
Implement business objects and utility classes.	7.4	1
Implement session state, view state, control state, cookies, cache, or application state.	7.5	7, 9
Handle events and control page flow.	7.6	1, 2
Implement the generic handler.	7.7	12

This page intentionally left blank

A

AcceptChanges method, 119–120

AccessDataSource control, 51, 137

Accessibility

 data input, 252

 data output, 252

 defined, 233

 keyboard operations, 253–254

 site navigation, 252, 253–254

 testing, 254–256

 textual alternatives for non-text elements, 252

 UI elements, 252–253

 user agent independence, 253

 web applications, 251–256

ActiveX Data Objects. *See ADO*

ADO, 102–103

ADO.NET

 command objects, 113–115

 connected classes, 106–115

 connection objects, 106–112

 DataAdapter events, 134–136

 DataAdapter object, 115, 124–134

 data manipulation, 118–121

 data providers, 103–105

 DataReader object, 115, 123–124

 DataSet object, 121–122

 data source controls, 136–140

 DataTable object, 115–118

 disconnected classes, 115–122

 LINQ to SQL, 155

AdRotator control

 cache substitution, 265

 data binding, 56

AJAX

 in ASP.NET applications, 169–183

 at client side, 169, 176–178

 communication architecture, 164–165

 defined, 162

 DOM, 163

 events, 178–179

 how it works, 163–164

 HTTP, 163

 JSON, 165–168

 at server side, 168, 169–176

 System.Web.UI.Control, 168

 unhandled exceptions, 367

 Windows Communication Foundation (WCF), 165

 web services, 179–183, 306–307

AJAX architecture

 JSON, 165

 page methods, 165

 service proxies, 165

 web services, 165

AJAX at client side

 ClientScriptManager class, 176–177

 script, adding dynamically, 176–177

 script block, defining, 176

 ScriptManager control, 177–178

 ScriptReference object, 177–178

AJAX events

 Application class, 178

 client, 178

 page, 179

 PageRequestManager class, 178–179

AJAX server-side controls

 ScriptManager, 169

 ScriptManagerProxy, 169

 Timer, 175–176

 UpdatePanel, 169–173

 UpdateProgress, 173–174

AJAX web services

 creating, 180–182, 306–307

 server-side code, calling, 182–183

 WebMethod attribute, 182–183

 WebMethods, 180–183

Application data caching

 Add() method, 266–267

 Cache class, 266

 defined, 266

 dependencies, 267–268

 Get() method, 267

 Insert() method, 267

 using, 268–272

Application state

 HttpApplicationState class, 228

 state management, 228

ASP.NET 3.5

 and AJAX, 169–183

 assemblies, 8–9

 client-side AJAX support, 164, 169, 176–178

 components, 2–4

 development structure, 7

 events and methods, 7

 extensions, 318–321

 features of, 2, 318

 HTTP methods, 4–5

 Internet Information Services (IIS), 3

 ISAPI DLLs, 3–4

 life cycle stages, 5–6

 project types, 7–8

 server-side AJAX support, 164, 168

 URL mapping, static, 219

 web browser, 3

 web server, 2–3

ASP.NET AJAX script libraries, 169

ASP.NET AJAX Control Toolkit, 179

- ASP.NET components
 - Internet Information Services (IIS), 3
 - ISAPI DLLs, 3–4
 - web browser, 3
 - web server, 2–3
- ASP.NET Web Services
 - classes, 303
 - creating, 303–305
 - ScriptServiceAttribute class, 304
 - using, 305–310
 - WebMethodAttribute class, 304
 - WebService class, 303–304
- Assemblies
 - defined, 8
 - external, 8
 - local, 8
 - private, 8
 - references, adding, 8–9
 - shared, 8
 - System.Web.dll, 8
- Asynchronous, defined, 163
- Asynchronous JavaScript and XML. *See* AJAX
- Authenticate event, 281
- Authentication
 - defined, 272
 - forms, 273–275
 - membership providers, 275–277
 - modes, 272–273
 - password, 273
 - security, 272–277
 - user identities, 275
 - web services security, 309–310
- Windows, 272–273
- Authorization
 - defined, 292
 - file, 292
 - impersonation, 296–298
 - location element, 294
 - role management, 294–296
 - security, 291–299
 - URL, 292
 - web services security, 309–310
- B**
 - BaseCompareValidator class, 191
 - BaseValidator class
 - properties, 189–190
 - Validate method, 190
 - Breakpoints, defined, 360
 - BulletedList control, 55
 - Button control, properties, 36
- C**
 - Cache class
 - methods, 266–267
 - properties, 266
 - CacheDependency class, 267–268
 - Caching
 - application data, 266–272
 - fragment, 266
 - output, 260–266
 - Calendar control, 42
 - Callback, 163, 265
 - Cascading style sheets (CSS), 18
 - ChangePassword control, 284–285
 - CheckBox control, 36–37
 - CheckBoxList control, 54, 55
 - Choice element, 331–332
 - Client, defined, 4
 - ClientScriptManager class, 176–177
 - Command control, 339–341
 - Command-line tool
 - compilation, 24–27
 - LINQ, 156–157
 - Object relational (O/R) mapping, 156–157
 - Command objects
 - CommandType property values, 114
 - DbParameter object, 114–115
 - SqlCommand class, 113–114
 - CompareValidator class, properties, 193
 - CompareValidator control, 192–193, 199
 - Comparison-based filter control, 331
 - Compilation
 - command-line tool, 24–27
 - defined, 357
 - dynamic, 24
 - model, 22–27
 - sequence, 23
 - Composite control, defined, 88
 - Configuration
 - machine, 21
 - .NET, 21
 - section handlers, 21–22
 - web, 22
 - Connected classes
 - Command objects, 113–115
 - Connection objects, 106–115
 - DataAdapter objects, 115
 - DataReader objects, 115
 - Connection pooling, 111–112
 - Connection strings
 - OBDC, 107
 - OLEDB, 107–108
 - SQL Server, 108–110
 - web.config file, 110–111, 140
 - Container control, 339
 - Control adapters, 346–347
 - Control class, properties, 222–223
 - Control directive, 73
 - Controls. *See* individual control types
 - Control state, 224
 - Cookieless sessions, 348
 - Cookies
 - deleting, 227
 - HttpCookie class, 225
 - HttpResponse object, 226
 - HttpRequest object, 226
 - state management, 225–227
 - Copy Web tool, 380–381
 - CreateUserWizard control, 285–291
 - CultureInfo class
 - globalization, 235–237
 - methods, 237

- properties, 236
- using, 256
- cursor, defined, 113
- Custom control, defined, 83
- CustomValidator control
 - description, 199
 - fields, 197
 - handling, 194–195
 - OnServerValidate() method, 194
 - ServerValidate event, 194
 - ValidateEmptyText property, 194
- D**
- DataAdapter events
 - FillError, 135–136
 - RowUpdating and RowUpdated, 134–135
 - Status property values, 135
- DataAdapter object
 - connected architecture, 115
 - constraints, adding into a data set, 127–128
 - data manipulation, 124–125
 - data sources, updating, 131
 - DataTable and DataColumn objects, mapping, 128–130
 - events, 134–136
 - Fill method, 125, 127–128, 130–131
 - FillSchema method, 125, 128
 - methods, 125
 - multiple, populating a data set, 125–126
 - paging to display result sets, 130–131
 - parameters, 126–127
 - properties, 124
 - Update method, 125, 131
 - UpdateRowSource property, 131–134
 - working with, 115
- Database, defined, 102
- DataBinder class, 52–53
- Data-bound controls
 - AdRotator, 56
 - binding data to, 50
 - categories of, 50
 - collection, creating, 49–50
 - composite, 50, 55
 - DataBinder class, 52–53
 - DataGridView, 62
 - DataTable, 62–64
 - DataPager, 66–69
 - DataSource objects, 51–52
 - DetailsView, 56–58
 - FormView, 58–60
 - GridView, 55–56
 - GUI-based data source, 51–52
 - hierarchical, 50, 55
 - List, 54–55
 - ListView, 51
 - Menu, 61
 - Repeater, 64–66
 - simple, 50
 - Template, 53
 - TreeView, 60–61
- Data caching. *See* Application data caching
- DataTable object, 116, 128–130
- DataGridView control, 62
- DownList control, 62–64
- Data manipulation
 - AcceptChanges method, 119–120
 - ADO.NET, 118–121
 - DataAdapter object, 124–125
 - duplication, 120–121
 - modification, 119–120
 - version identification, 118
- DataPager control
 - data binding, 66–69
 - and ListView control, 67–68
 - pager field types, 67
 - read-only properties, 67
- Data providers, 103–105
- DataReader object
 - closing, 124
 - connected architecture, 115
 - data retrieval, 123–124
 - working with, 115
- DataSet object, 121–122
- Data source controls
 - ADO.NET, 136–140
 - AccessDataSource, 51, 137
 - built-in, 136–137
 - creating, 137–138
 - EntityDataSource, 136
 - LinqDataSource, 52, 136
 - ObjectDataSource, 52, 136–137
 - SiteMapDataSource, 52, 137
 - SqlConnection, 139
 - SqlDataSource, 51, 136
 - web.config file connection string, reading, 140
 - XmlDataSource, 52, 137
- DataSource objects, 51–52
- DataTable object
 - ADO.NET disconnected classes, 115–118
 - DataRow objects, 117–118
 - GUID as primary key, 117
 - Load method, 117
 - mapping, 128–130
 - primary key, assigning, 117
 - rows, inserting, 117–118
- DbParameter object, 114–115
- Debugging
 - applications, steps to, 359–360
 - deployment issues, 361–362
 - remotely, 362
 - version conflicts, 362
 - with Visual Studio, 358–361, 367
- Default skin, 18
- Deployment
 - Copy Web tool, 380–381
 - precompilation, 381–383
 - properties and conditions, 377–380
 - Web Setup Project, 375–377, 385
- DetailsView control
 - data binding, 56–58
 - versus FormView control, 58
- Development structure, 7
- Device-specific rendering
 - choice element, 331–332
 - device filters, 331

Device-specific rendering (*continued*)

- <device-specific><choice> construct, 331
- filter property, 331
- methods, 330
- mobile applications, 330–332

Directives

- Control, 73
- Master, 13
- OutputCache, 298–299
- Page, 19
- Reference, 75
- Register, 74–75

Disconnected classes

- Data manipulation, 118–121
- DataSet object, 121–122
- DataTable object, 115–118

Document Object Model. *See* DOM

DOM

- AJAX, 163
- defined, 145
- XML document, parsing, 148–149
- XML document, searching, 150

Dynamic Data, 318, 319

Dynamic-link libraries (DLLs), defined, 3

E

Emulator, 332–333

EntityDataSource control, 136

Error handling

- error pages, custom, 363–365
- handlers, 367–371
- try/catch blocks, 362–363
- unhandled exceptions, 366–367

Evaluator-delegate-based filter, 331

Events

- AJAX, 178–179
- application, 7
- authenticate, 281
- DataAdapter object, 134–136
- Logginin, 281
- ServerValidate, 194
- TraceFinished, 356–357
- User control, 76–78

Explicit localization, 243

Extensible Markup Language (XML). *See* XMLExtensible Stylesheet Language (XSL). *See* XSL**F**

File configuration and compilation

- compilation model, 22–27
- web applications, 20–22

FileUpload control, 48–49

FillError, 135–136

Fill method, 125, 127–128, 130–131

FillSchema method, 125, 128

Filter property, 331

Foreign key, defined, 121

FormView control

- data binding, 58–60
- versus DetailsView control, 58

Fragment caching, defined, 266

G

Global assembly cache (GAC), defined, 8

Globalization

- and applications, creating, 245–251
- best practices, 244–245
- culture, 235–239
- culture, invariant, 236
- CultureInfo class, 235–237
- defined, 233
- demonstrating, 245–251
- implementing, 234–242
- internationalization, 234
- localizable resources, 234
- resource files, 239–242
- System.Globalization namespace, 235

Global resources, 239, 243–244

Globally unique identifier (GUID), 117

GridView control, 55–56

H

Handlers

- built-in, 369
- custom, 367–369, 370–371
- defined, 367
- generic, 369–370
- IHttpHandler, 368–369
- and session state, 369

HiddenField control, 221–222, 229–230

HotSpot control, 43

HTML controls

- properties, 33
- versus web server controls, 32–34

HttpApplication class, 14–15

HttpApplicationState class, 228

HttpCachePolicy class, 261

HttpContext class, 16

HttpCookie class, 225

HTTP methods

- HTTP request, 4–5
- HTTP response, 5
- HTTP troubleshooting tools, 5

HTTP request, defined, 2

HttpRequest class, 15–16

HttpRequest object, 226

HTTP response, defined, 5

HttpResponse object, 226

HttpRuntime class, 16–17

HttpServerUtility class, 17

HyperLink control, 207

Hypertext Transfer Protocol. *See* HTTP**I**

IHttpHandler, 368–369

ImageButton control, 43

ImageMap control, 43

Impersonation

- authentication, 296–298
- defined, 272

Implicit localization, 242–243

In-line coding versus code-behind programming, 20

Input validation

- BaseCompareValidator class, 191
- BaseValidator class, 189–190
 - client-side support, 198–199
 - CompareValidator control, 192–193, 199
 - CustomValidator control, 194–195, 197, 199
 - defined, 187
 - examples, 187
 - implementing, 187–198
 - mobile controls, 335–336
 - Page object support, 195–196
 - programming, 186–199
 - RangeValidator control, 192, 196, 199
 - RegularExpressionValidator control, 193, 196, 199
 - RequiredFieldValidator control, 192, 196
 - server-side, implementing, 196–198
 - server-side object model, 188–189
 - server-side support, 188
 - SQL injection, 187
- Internet Information Server, defined, 2
- Internet Information Services (IIS), 3
- Internet service application programming interface. *See* ISAPI
- Intrinsic objects
 - HttpApplication class, 14–15
 - HttpContext class, 16
 - HttpRequest class, 15–16
 - HttpRuntime class, 16–17
 - HttpServerUtility class, 17
- InvariantCulture, 236
- ISAPI
 - defined, 3
 - DLLs, 3–4
- J**
 - JavaScript Object Notation. *See* JSON
- JSON
 - and AJAX, 165–168
 - array, using as, 168
 - defined, 165
 - literals, 165
 - name-value pair, using as, 168
 - Object Literal Notation, 165
- L**
 - Label control, properties, 35
 - Language-integrated query. *See* LINQ
 - Life cycle stages
 - ASP.NET applications, 5–6
 - web page, 31–32
 - LINQ
 - code, writing, 155
 - command-line tool, 156–157
 - defined, 144
 - entities, creating, 154–155
 - entities, mapping, 157
 - O/R mapping, 154, 155–157
 - queries, 157–158
 - SqlMetal.exe, 156–157
 - Visual Studio Designer Tool, 155
 - LinqDataSource control, 52, 136
 - LINQ to SQL
 - accessing data, 154–155
 - ADO.NET, 155
- insert, update, and delete, 158
- queries, 157–158
- LINQ to XML, 155
- List control
 - data binding, 54–55
 - mobile applications, 341–343
- ListView control
 - data binding, 51
 - and DataPager control, 67–68
- Literal, 165
- Literal control, 207, 243
- Localization
 - and applications, creating, 245–251
 - best practices, 244–245
 - defined, 233
 - demonstrating, 245–251
 - explicit, 243, 248
 - implementing, 234–242
 - implicit, 242–243
- Localize control, 243
- Local resources, 239
- loggedin event, 281
- Login controls, 278–291
 - Authenticate event, 281
 - ChangePassword, 284–285
 - CreateUserWizard, 285–291
 - loggedin event, 281
 - Login, 278–281
 - LoginName, 281
 - LoginStatus, 281–282
 - LoginView, 282
 - PasswordRecovery, 283–284
 - security, 278–291
- M**
 - Machine.config file, 21
 - Master directive, 13
 - Master pages, 12–13
 - application level, 13
 - defined, 12
 - folder level, 13
 - nested, 13
 - page level, 12
 - Membership provider, defined, 275
 - Menu class
 - methods, 210
 - properties, 209
 - Menu control
 - data binding, 61
 - SiteMap class, linking to, 214
 - Site navigation, 207–211
 - MenuItem class
 - constructors, 210
 - properties, 210–211
 - Microsoft Mobile Internet Toolkit (MMIT), 326
 - Mobile applications
 - choice element, 331–332
 - control adapters, 346–347
 - controls, 335–347
 - developing, 327–330
 - device filters, 331
 - <Device-Specific><Choice> construct, 331

Mobile applications (*continued*)
 device-specific rendering, 330–332
 emulator, 332–333
 filter property, 331
 local, 329
 objects, 327
 state management, 347–349
 types, 329
 web, creating, 329–330
 web, viewing and testing, 332–333
 web-based, 329
 web browser, 332
 web forms, 333–335
 workflow, 327–329

Mobile controls
 Command, 339–341
 Container, 339
 defined, 327
 input validation, 335–336
 List, 341–343
 PhoneCall, 343–344
 RangeValidator, 344–345
 StyleSheet, 344
 types, 335–336
 UI, 336
 utility, 335
 working with, 336–339

Model View Controller, 318, 319–320

Multipurpose Internet Mail Extension (MIME), defined, 5

MultiView control, 49

N
 Named skin, 18
 Navigation controls. *See also* Site navigation
 linking, 213–218
 Menu, 207–211
 site management, 205–218
 SiteMapPath, 205–207
 TreeView, 211–213
 .NET configuration, 21
 NeutralCutures, 236

O
 ObjectDataSource control, 52, 136–137
 Object relational (O/R) mapping
 code editor, 157
 command-line tool, 156–157
 to entities, 157
 generating, 155–156
 SqlMetal.exe, 156–157
 Visual Studio Designer Tool, 155
 OutputCache directive, 298–299
 Output caching
 cached pages, invalidating, 262–263
 cache substitution, 265
 configuring, 264–265
 defined, 260
 file and cache key dependencies, 263–264
 HttpCachePolicy class, 261
 implementing, declaratively, 260–261
 implementing, programmatically, 261–262

tasks, 262
 web.config file, 264

P
 Page directive, 19
 Page object, 195–196
 PageRequestManager class, 178–179
 Panel control, 44
 PasswordRecovery control, 283–284
 Performance
 data caching, 266–272
 fragment caching, 266
 output caching, 260–266
 Personalization, 93
 PhoneCall control, 343–344
 Postback, defined, 163
 Precompilation
 deployment, 382
 methods, 381
 options, 381–382
 performance, 381–382
 steps to, 382–383
 Primary key, defined, 116
 Profile feature, 229
 Profile properties, 229
 Programming
 input validation, 186–199
 site management, 200–220
 state management, 220–229
 Project types, 7–8
 Publishing
 application health, monitoring, 384
 file system project, using, 383–385
 FTP project, using, 384
 HTTP project, using, 383

Q
 Queries
 LINQ, 157–158
 LINQ to SQL, 157–158
 Query string
 Request object, 225
 state management, 224–225

R
 RadioButton control, 37–38
 RadioButtonList control, 55
 RangeValidator control
 input validation, 192, 196, 199
 mobile controls, 344–346
 Rapid Application Development (RAD), 52
 Reference directive, 75
 Register directive, 74–75
 RegularExpressionValidator control, 193, 196, 199
 Remapping, defined, 220
 Repeater control
 data binding, 64–66
 templates, 64
 Representational State Transfer. *See* REST
 Request, defined, 32
 RequiredFieldValidator control, 192, 196
 Resource files

- creating, 240–242
 - localization, 242–243
 - types, 239
- Response, defined, 32
 - REST, and AJAX, 165
- Resultset, defined, 115
 - Role management, 294–296
- RowUpdated, 134–135
 - RowUpdating, 134–135

- S**
 - ScriptManager control, 169, 177–178
 - ScriptManagerProxy control, 169
 - ScriptReference object, 177–178
 - ScriptServiceAttribute class, 304
 - Security
 - authentication, 272–277
 - authorization, 291–299
 - Login controls, 278–291
 - Serialization, defined, 168
 - Server controls, defined, 31. *See also* Web server controls
 - Service-Oriented Architecture, defined, 311
 - Sessions
 - HttpSessionState class, 227
 - state management, 227–228
 - variables, 228
 - Silverlight, 318, 320–321
 - applications, creating, 321
 - architecture, 320
 - controls, 321
 - Simple Object Access Protocol. *See* SOAP
 - Site management
 - Menu control, 207–211
 - navigation, 200
 - navigation controls, 205–218
 - programming, 200–220
 - SiteMap class, 205
 - SiteMapPath control, 205–207
 - site maps, 201–205
 - TreeView control, 211–213
 - URL mapping, 218–220
 - web.sitemap file, creating, 201–203
 - SiteMap class
 - Menu control, linking to, 214
 - properties, 205
 - SiteMapDataSource class
 - properties, 213–214
 - TreeView control, associating, 215
 - SiteMapDataSource control, 52, 137
 - SiteMapPath class, properties, 206–207
 - SiteMapPath control
 - bread-crumb trail, 217
 - node types, 205
 - site navigation, 205–207
 - Site maps
 - multiple, 203–204
 - site management, 201–205
 - SiteMap class, 205
 - web.sitemap file, creating, 201–204
 - Site navigation. *See also* Navigation controls
 - accessibility, 252, 253–254
 - defined, 200
 - Menu control, 207–211
 - TreeView control, 211–213
 - Skins, types, 18
 - SOAP
 - and AJAX, 165
 - defined, 302
 - headers, to secure web services, 307–309
 - Solution files, 8
 - SpecificCultures, 236
 - SqlCommand class, 113–114
 - SqlConnection, 139
 - SqlDataSource control, 51, 136
 - SQL injection, 187
 - SqlMetal.exe, 156–157
 - State management
 - application state, 228
 - client-side, 221
 - client-side versus server-side, 221
 - control state, 224
 - cookies, 225–227
 - defined, 220
 - HiddenField control, 221–222, 229–230
 - hidden fields, 221–222
 - mobile applications, 347–349
 - profile properties, 229
 - programming, 220–229
 - query string, 224–225
 - server-side, 221
 - sessions, 227–228
 - view state, 222–224
 - StyleSheet control, 344
 - Substitution control, 265
 - System.Diagnostics namespace, 352, 353, 354

 - T**
 - Table, defined, 106
 - TableCell control, 41–42
 - Table control, 41–42
 - TableRow control, 41–42
 - Telnet, 24–25
 - Template control, 53
 - Templated user controls
 - creating, 79–81
 - defined, 78
 - using, 81–82
 - TextBox control, properties, 35–36
 - Theme graphics, 18–19
 - global level, 19
 - web application level, 18
 - Themes, 18–19
 - Timer control, 175–176
 - Trace class, 352, 353
 - TraceFinished event, 356–357
 - Tracing
 - application, 352–353, 354–355
 - defined, 351
 - enabling, 353–354
 - page, 352–353
 - statements, adding, 355–356
 - TreeNode class
 - constructors, 212
 - methods, 213

- TreeNode class (*continued*)
 properties, 213
- TreeView class
 methods, 212
 properties, 211–212
- TreeView control
 data binding, 60–61
 node types, 60
 site navigation, 211–213
- Troubleshooting
 defined, 351
 System.Diagnostics namespace, 352, 353, 354
 Trace class, 352
 tracing, 352–357
- U**
- UI control, 336
- UI structural controls, 93
- Unhandled exceptions
 AJAX, 367
 defined, 366
 handling, steps to, 366–367
- Update method, 125, 131
- UpdatePanel control
 AJAX, 169–173
 partial page rendering, 170, 173
 postback, 170
 triggers, 172
- UpdateProgress control
 AJAX, 173–175
 properties, 173
- UpdateRowSource property, 131–134
- URL mapping
 custom, 220
 site management, 218–220
 static, 219–220
- URL remapping, 220
- User controls
 adding to a web page, 73
 Control directive, 73
 creating, 73–75, 97–98
 data, accessing, 75–76
 defined, 73
 events, raising, 76–78
 loading, 76
 positioning, 76
 templated, 78–82
 versus custom controls, 92
 versus web pages, 73
- User interface (UI), accessibility elements, 252–253
- Users, defined, 3
- V**
- Validation controls
 CompareValidator, 192–193, 199
 CustomValidator, 194–195, 199
 RangeValidator, 192, 199
 RegularExpressionValidator, 193
 RequiredFieldValidator, 192
- ValidationSummary class, properties, 195
- View state
 Control class, 222–223
- defined, 32
 encryption, 223
 MaxPageStateFieldLength, 223
 state management, 222–224
- ViewState control, 348–349
- Visual Studio
 auto window, 361
 call stack window, 361
 debugging with, 358–361
 Designer tool, 155
 immediate window, 360
 local window, 361
 watch window, 361
- W**
- Web application projects, versus Web site projects, 9–11
- Web applications
 accessibility, 251–256
 building, 25–27
 compilation model, 22–25
 compiling, 381–383
 configuring, 20–22
 debugging, 357–362
 deployment, 375–381
 error logging, 362–371
 health, monitoring, 384–385
 mobile, 326–335
 performance, 260–271
 publishing, 383–382
 security, 272–298
 troubleshooting, 351–357
- Web browser, 2, 3, 332
- Web.config file
 connection string, reading from, 140
 parameters, setting, 22
- Web farm, defined, 375
- Web forms, mobile, 333–335
- WebMethod attribute, 182–183
- WebMethodAttribute class, 304
- WebMethods, 180–183
- Web page
 adding, 10–11, 25–27
 “bread-crumb” trail, 205
 converting to a user control, 75
 creating, 38–41
 defined, 3
 life cycle, 31–32
- Web page structure
 code, 19
 directives, 19
 in-line coding versus code-behind programming, 20
 layout, 19
- WebPartManager class, 96–98
- Web Parts
 application development, 96
 built-in, 97
 defined, 92
 pages, developing, 94–95
 personalization, 93
 UI controls, 93
- Web Parts controls
 components, 93

- creating, 93–94
- developing, 93–94
- Web server**, 2–3
- Web server controls**
 - Button, 36
 - Calendar, 42
 - CheckBox, 36–37
 - data-bound, 49–69
 - defined, 31
 - FileUpload, 48–49
 - HotSpot, 43
 - versus HTML controls, 32–34
 - ImageButton, 43
 - ImageMap, 43
 - images, displaying, 43
 - Label, 35
 - MultiView, 49
 - Panel, 44
 - properties, 34
 - RadioButton, 37–38
 - specialized, 41–49
 - standard, 35–41
 - tables, displaying data, 41–42
 - TextBox, 35–36
 - Wizard, 44–48
- Web server controls, custom**
 - adding to a web page, 84
 - adding to the toolbox, 86
 - composite, creating, 88
 - creating, from the WebControl class, 85–86
 - creating, using another control, 83
 - custom designers, 87–88
 - defined, 83
 - individualizing, 86–87
 - templated, creating, 88–91
 - templated, using, 91–92
 - versus user controls, 92
- Web server controls, specialized**
 - Calendar, 42
 - FileUpload, 48–49
 - HotSpot, 43
 - ImageButton, 43
 - ImageMap, 43
 - MultiView, 49
 - Panel, 44
 - Table, 41–42
 - TableCell, 41–42
 - TableRow, 41–42
 - tables, displaying data, 41–42
 - Wizard, 44–48
- Web server controls, standard**
 - Button, 36
 - CheckBox, 36–37
 - Label, 35
 - RadioButton, 37–38
 - TextBox, 35–36
- WebService class**, 303–304
- Web services**
 - AJAX, creating, 180–182, 306–307
 - ASP.NET, 302–310
 - calling, 306
 - defined, 165
 - referencing, 305–306
 - ScriptServiceAttribute class, 304
 - securing, with authorization and authentication, 309
 - securing, with SOAP headers, 307–309
 - security, configuring, 309–310
 - WebMethodAttribute class, 182–183, 304
 - WebServiceAttribute class, 303
 - WebService class, 303–304
 - XML, creating, 304–305, 322–324
- Web Services Description Language (WSDL)**, defined, 302
- Web Setup Project**
 - building, steps to, 376–377
 - deploying, steps to, 385
 - deployment, application, 375–379
 - options, 377
 - properties, 376
- Web site**
 - appearance, 9–20
 - cascading style sheets (CSS), 18
 - creating, 10–11
 - file-based, 7
 - FTP-based, 7
 - HTTP-based, 7
 - intrinsic objects, 14–17
 - layout, 9–20
 - master pages, 12–13
 - page structure, 19–20
 - projects versus web application projects, 9–11
 - remote HTTP-based, 7
 - skins, 18
 - theme graphics, 18–19
 - themes, 18
 - types, 7
- Web.sitemap file**
 - creating, 201–204
 - multiple, 203
- WebZones**, 96–98
- Windows authentication**, 272–273
- Windows Communication Foundation (WCF)**
 - and AJAX, 165
 - applications, hosting, 318
 - defined, 311
 - elements, 311–312
 - message exchange patterns, 312
 - need for, 311
 - service, calling asynchronously, 317–318
 - service, calling synchronously, 315–317
 - service, creating, 312–315
 - service, using, 315
- Windows Installer**, defined, 376
- Wireless Markup Language (WML)**, 335
- Wizard control**, 44–48
 - classes, related, 48
 - properties, 44–45
- WriteSubstitution() method**, 265

X**XML**

- defined, 144
- documents, 147–154
- namespaces and classes, 145–147
- web services, 304–306

- XmlConvert class, 146
- XmlDataDocument class, 145–146
- XmlDataSource control, 52, 137
- XmlDocument class, 145–146
- XML documents
 - creating, 147–148, 159
 - modifying, 153
 - parsing with DOM, 148–149
 - reading using XmlTextReader class, 152–153
 - searching with DOM, 150
 - validating, 153–154
 - writing using XmlTextWriter class, 151–152
- XPath, 150–151
- XML namespaces and classes
 - XmlConvert class, 146
 - XmlDataDocument class, 145–146
 - XmlDocument class, 145–146
 - XmlNodeReader class, 146
 - XmlReader class, 146
 - XmlTextReader class, 145, 147, 152–153
 - XmlTextWriter class, 145, 147, 151–152
- XPathDocument class, 146, 150–151
- XPathNavigator class, 146, 150–151
- XslTransform class, 146
- XmlNodeReader class, 146
- XML processing
 - in-memory, 145
 - stream-based, 145
- XmlReader class, 146
- XmlTextReader class, 145, 147, 152–153
- XmlTextWriter class, 145, 147, 151–152
- Xml Web Services
 - creating, 304–305, 322–324
 - defined, 303
- XPathDocument class, 146, 150–151
- XPathNavigator class, 146, 150–151
- XSL, defined, 145
- XslTransform class, 146

Z

- Zones, defined, 93, 96–97