

ASP.NET 2.0 Beta Preview

Written and tested for final release of **ASP.NET 2.0 Beta 1**

Bill Evjen



Updates, source code, and Wrox technical support at www.wrox.com

ASP.NET 2.0 Beta Preview

Bill Evjen



Wiley Publishing, Inc.

ASP.NET 2.0 Beta Preview

ASP.NET 2.0 Beta Preview

Bill Evjen



Wiley Publishing, Inc.

ASP.NET 2.0 Beta Preview

Published by
Wiley Publishing, Inc.
10475 Crosspoint Boulevard
Indianapolis, IN 46256
www.wiley.com

Copyright © 2004 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 0-7645-7286-5

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/ST/QX/QU/IN

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, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, E-Mail: brandreview@wiley.com.

LIMIT OF LIABILITY/DISCLAIMER OF WARRANTY: THE PUBLISHER AND THE AUTHOR MAKE NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS WORK AND SPECIFICALLY DISCLAIM ALL WARRANTIES, INCLUDING WITHOUT LIMITATION WARRANTIES OF FITNESS FOR A PARTICULAR PURPOSE. NO WARRANTY MAY BE CREATED OR EXTENDED BY SALES OR PROMOTIONAL MATERIALS. THE ADVICE AND STRATEGIES CONTAINED HEREIN MAY NOT BE SUITABLE FOR EVERY SITUATION. THIS WORK IS SOLD WITH THE UNDERSTANDING THAT THE PUBLISHER IS NOT ENGAGED IN RENDERING LEGAL, ACCOUNTING, OR OTHER PROFESSIONAL SERVICES. IF PROFESSIONAL ASSISTANCE IS REQUIRED, THE SERVICES OF A COMPETENT PROFESSIONAL PERSON SHOULD BE SOUGHT. NEITHER THE PUBLISHER NOR THE AUTHOR SHALL BE LIABLE FOR DAMAGES ARISING HEREFROM. THE FACT THAT AN ORGANIZATION OR WEBSITE IS REFERRED TO IN THIS WORK AS A CITATION AND/OR A POTENTIAL SOURCE OF FURTHER INFORMATION DOES NOT MEAN THAT THE AUTHOR OR THE PUBLISHER ENDORSES THE INFORMATION THE ORGANIZATION OR WEBSITE MAY PROVIDE OR RECOMMENDATIONS IT MAY MAKE. FURTHER, READERS SHOULD BE AWARE THAT INTERNET WEBSITES LISTED IN THIS WORK MAY HAVE CHANGED OR DISAPPEARED BETWEEN WHEN THIS WORK WAS WRITTEN AND WHEN IT IS READ.

For general information on our other products and services or to obtain technical support, please contact our Customer Care Department within the U.S. at (800) 762-2974, outside the U.S. at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Evjen, Bill.

ASP.NET 2.0 Beta Preview / Bill Evjen.

p. cm.

Includes index.

ISBN 0-7645-7286-5 (paper/website)

1. Active server pages. 2. Web sites--Design. I. Title.

TK5105.8885.A26E95 2004

005.2'76--dc22

2004011609

Trademarks: Wiley, the Wiley Publishing logo, Wrox, the Wrox logo, Programmer to Programmer and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

About the Author

Bill Evjen is an active proponent of .NET technologies and community-based learning initiatives for .NET. He has been actively involved with .NET since the first bits were released in 2000. In the same year, Bill founded the St. Louis .NET User Group (<http://www.stlnet.org>), one of the world's first .NET user groups. Bill is also the founder and executive director of the International .NET Association (<http://www.ineta.org>), which represents more than 200,000 members worldwide.

Based in St. Louis, Missouri, USA, Bill is an acclaimed author and speaker on ASP.NET and XML Web services. He has written or coauthored *Professional C#*, 3rd Edition and *Professional VB.NET*, 3rd Edition (Wrox), *XML Web Services for ASP.NET*, *Web Services Enhancements: Understanding the WSE for Enterprise Applications*, *Visual Basic .NET Bible*, and *ASP.NET Professional Secrets* (all published by Wiley). In addition to writing, Bill is a speaker at numerous conferences including DevConnections, VSLive, and TechEd.

Bill is a Technical Director for Reuters, the international news and financial services company, and he travels the world speaking to major financial institutions about the future of the IT industry. He graduated from Western Washington University in Bellingham, Washington, with a Russian language degree. When he isn't tinkering on the computer, he can usually be found at his summer house in Toivakka, Finland. You can reach Bill at evjen@yahoo.com. He presently keeps his weblog at <http://www.geekswithblogs.net/evjen>.

Credits

Senior Acquisitions Editor

Jim Minatel

Development Editors

Brian Herrmann

Jodi Jensen

Production Editor

Gabrielle Nabi

Technical Editor

Devin Rader

Copy Editor

Mary Lagu

Editorial Manager

Mary Beth Wakefield

Vice President & Executive Group Publisher

Richard Swadley

Vice President and Publisher

Joseph B. Wikert

Project Coordinator

Bill Ramsey

Graphics and Production Specialists

Sean Decker

Carrie Foster

Jennifer Heleine

Lynsey Osborn

Media Development Specialist

Travis Silvers

Proofreading

Kim Cofer

Indexing

Johnna VanHoose Dinse

To Tuija

Contents

Introduction	xvii
Acknowledgments	xxv
Chapter 1: Introduction to ASP.NET 2.0	1
A Little Bit of History	2
The Goals of ASP.NET 2.0	2
Developer productivity	3
Administration and management	6
Performance and scalability	7
Device-specific code generation	8
Additional New Features of ASP.NET 2.0	9
New developer infrastructures	9
New compilation system	15
Additions to the page framework	15
New objects for accessing data	18
New server controls	18
A New IDE for Building ASP.NET 2.0 Pages	19
Summary	21
Chapter 2: Visual Studio 2005	23
The Document Window	23
Views in the Document Window	23
The tag navigator	25
Page tabs	25
Code change status notifications	26
Error notifications and assistance	27
The Toolbox	29
The Solution Explorer	31
The Server Explorer	33
The Properties Window	33
Lost Windows	34
Other Common Visual Studio Activities	34
Creating new projects	35
Making references to other objects	35

Contents

Using smart tags	37
Saving and importing Visual Studio settings	38
Summary	40
Chapter 3: Application and Page Frameworks	41
Application Location Options	41
Built-in Web server	41
IIS	43
FTP	44
Web site requiring FrontPage Extensions	45
The ASP.NET Page Structure Options	45
Inline coding	47
New code-behind model	49
New Page Directives	51
New attributes	51
New directives	52
New Page Events	53
Cross-Page Posting	54
New Application Folders	61
\Code folder	61
\Themes folder	66
\Resources folder	66
Compilation	69
Summary	73
Chapter 4: New Ways to Handle Data	75
The New Data Source Controls	75
The Data-Bound Server Controls	76
The SqlDataSource and GridView Controls	77
Reading data	77
Applying paging in the GridView	79
Sorting rows in the GridView control	82
Defining bound columns in the GridView control	83
Enabling the editing of rows in the GridView control	87
Deleting data from the GridView	92
Dealing with other column types in the GridView	93
The AccessDataSource and DetailsView Controls	96
Selecting which fields to display in the DetailsView control	101
Using the GridView and DetailsView together	103
Updating, inserting, and deleting rows	105

XmlDataSource Control	109
ObjectDataSource Control	114
SiteMapDataSource Control	116
DataSetDataSource Control	117
Visual Studio 2005	118
Connection Strings	123
Summary	124
 Chapter 5: Site Navigation	 127
 Site Maps	 128
SiteMapPath Server Control	130
The PathSeparator property	132
The PathDirection property	134
The ParentLevelsDisplayed property	134
The ShowToolTips property	135
The SiteMapPath control's child elements	135
TreeView Server Control	136
Identifying the TreeView control's built-in styles	138
Examining the parts of the TreeView control	139
Binding the TreeView control to an XML file	140
Selecting multiple options in a TreeView	142
Specifying custom icons in the TreeView control	145
Specifying lines used to connect nodes	147
Working with the TreeView control programmatically	150
Menu Server Control	156
Applying different styles to the Menu control	157
Menu Events	163
Binding the Menu control to an XML file	163
SiteMap Data Provider	165
SiteMapViewType	165
StartingNodeType	166
SiteMap API	168
Summary	171
 Chapter 6: Working with Master Pages	 173
 Why Do You Need Master Pages?	 173
The Basics of Master Pages	176
Coding a Master Page	177
Coding a Content Page	180
Mixing page types and languages	184
Specifying which master page to use	186

Contents

Working with the page title	186
Working with controls and properties from the master page	187
Specifying Default Content in the Master Page	194
Nesting Master Pages	196
Container-Specific Master Pages	199
Event Ordering	200
Caching with Master Pages	201
Summary	201
 Chapter 7: Themes and Skins	 203
 Using ASP.NET 2.0 Packaged Themes	 203
Applying a theme to a single ASP.NET page	204
Applying a theme to an entire application	205
Applying a theme to all applications on a server	206
Removing themes from server controls	206
Removing themes from Web pages	207
Removing themes from applications	208
Creating Your Own Themes	208
Creating the proper folder structure	208
Creating a skin	209
Including CSS files in your themes	211
Having your themes include images	214
Defining Multiple Skin Options	218
Programmatically Working with Themes	220
Assigning the page's theme programmatically	220
Assigning a control's SkinID programmatically	220
Themes and Custom Controls	221
Summary	223
 Chapter 8: Membership and Role Management	 225
 Authentication	 226
Authorization	226
ASP.NET 2.0 Authentication	226
Setting up your Web site for membership	226
Adding users	229
Asking for credentials	236
Working with authenticated users	240
Showing the number of users online	242
Dealing with passwords	244

ASP.NET 2.0 Authorization	247
Using the LoginView server control	248
Setting up your Web site for role management	249
Adding and retrieving application roles	252
Deleting roles	255
Adding users to roles	256
Getting all the users of a particular role	256
Getting all the roles of a particular user	258
Removing users from roles	259
Checking users in roles	259
Using the Web Site Administration Tool	261
Summary	262
 Chapter 9: Personalization	 263
The Personalization Model	263
Creating Personalization Properties	264
Adding a simple personalization property	265
Using personalization properties	266
Adding a group of personalization properties	270
Using grouped personalization properties	271
Defining types for personalization properties	271
Using custom types	272
Providing default values	275
Making personalization properties read-only	275
Anonymous Personalization	275
Enabling anonymous identification of the end user	275
Working with anonymous identification events	278
Anonymous options for personalization properties	279
Migrating Anonymous Users	279
Personalization Providers	281
Working with the Access personalization provider	281
Working with the SQL Server personalization provider	282
Using multiple providers	289
Summary	290
 Chapter 10: Portal Frameworks and Web Parts	 291
Introducing Web Parts	291
Building Dynamic and Modular Web Sites	293
Introducing the WebPartManager control	293
Working with zone layouts	294

Contents

Understanding the WebPartZone control	298
Explaining the WebPartPageMenu control	301
Modifying zones	310
Working with Classes in the Portal Framework	317
Summary	322
 Chapter 11: SQL Cache Invalidation	 323
 Caching in ASP.NET 1.0/1.1	 323
Output caching	323
Partial page caching	324
Data caching using the Cache object	324
Cache dependencies	324
ASP.NET 2.0 unseals the CacheDependency class	325
Using the SQL Server Cache Dependency	325
Enabling databases for SQL Server cache invalidation	326
Enabling tables for SQL Server cache invalidation	327
Looking at SQL Server	327
Looking at the tables that are enabled	329
Disabling a table for SQL Server cache invalidation	329
Disabling a database for SQL Server cache invalidation	330
Configuring your ASP.NET Application	331
Testing SQL Server Cache Invalidation	332
Adding more than one table to a page	334
Attaching SQL Server cache dependencies to the Request object	334
Attaching SQL Server cache dependencies to the Cache object	335
Summary	339
 Chapter 12: Additional New Controls	 341
BulletedList Server Control	341
HiddenField Server Control	346
FileUpload Server Control	348
MultiView and View Server Controls	351
Wizard Server Control	355
Customizing the side navigation	357
Examining the AllowReturn attribute	357
Working with the StepType attribute	357
Adding a header to the Wizard control	358
Working with the Wizard's navigation system	359
Utilizing Wizard control events	360

DynamicImage Server Control	361
Working with images from disk	361
Resizing images	363
Displaying images from streams	364
ImageMap Server Control	366
Summary	368
 Chapter 13: Changes to ASP.NET 1.0 Controls	 369
Label Server Control	369
Button, LinkButton, and ImageButtonServer Controls	371
DropDownList, ListBox, CheckBoxLayout, and RadioButtonList Server Controls	372
Image Server Control	374
Table Server Control	374
Literal Server Control	376
AdRotator Server Control	376
Panel Server Control	380
Validation Server Controls	382
Summary	386
 Chapter 14: Administration and Management	 387
The MMC ASP.NET Snap-In	387
General	389
Custom Errors	390
Authorization	391
Authentication	393
Application	394
State Management	395
Advanced	397
ASP.NET Web Site Administration Tool	399
Home	401
Security	402
Profile	403
Application	404
Provider	405
Managing the Site Counter System	407
Summary	410
 Chapter 15: Visual Basic 8.0 and C# 2.0 Language Enhancements	 413
Overview of Changes	413
Generics	414

Contents

Iterators	419
Anonymous Methods	421
Operator Overloading	422
Partial Classes	422
Visual Basic XML Documentation	425
New Visual Basic Keywords	426
Continue	426
Using	428
My	428
Global	429
Summary	429
 Index	 431

Introduction

Simply put, ASP.NET 2.0 is an amazing release! When ASP.NET 1.0 was first introduced in 2000, many considered it a revolutionary leap forward in the area of Web application development. I believe ASP.NET 2.0 is just as exciting and revolutionary. Although the foundation of ASP.NET was laid with the release of ASP.NET 1.0, ASP.NET 2.0 builds upon this foundation by focusing on the area of developer productivity.

ASP.NET 2.0 brings with it a staggering number of new technologies that have been built into the ASP.NET framework. After reading this book, you will see just how busy the ASP.NET team has been in the last few years. The number of classes inside ASP.NET has more than doubled, and this release contains more than 40 new server controls!

This book covers these new built-in technologies; it not only introduces new topics, it also shows you examples of these new technologies in action. So sit back, pull up that keyboard, and let's have some fun!

What You Need for ASP.NET 2.0

You will probably install Visual Studio 2005 Beta 1 to work through the examples in this book. To work through *every* example in this book, you need

- ☐ Windows Server 2003, Windows 2000, or Windows XP
- ☐ Visual Studio 2005 Beta 1
- ☐ SQL Server 2000
- ☐ Microsoft Access

The nice thing is that you are not required to have IIS in order to work with ASP.NET 2.0 because this release of ASP.NET includes a built-in Web server. And if you don't have SQL Server, don't be alarmed. Many of the examples that use this database can be altered to work with Microsoft Access.

Who Is This Book For?

This book was written to introduce you to the new features and capabilities that ASP.NET 2.0 offers. This book is meant to be only an introduction to these new features. Therefore, I do not spend any time explaining the basics of ASP.NET and any functionality or capabilities that haven't changed between this release and the last release of ASP.NET.

This book is meant for the user who understands or has worked with ASP.NET 1.0 or 1.1. If you are brand new to Web application development, however, this book can help you get up to speed on the new features included in the upcoming release of ASP.NET—as long as you understand that the basics

Introduction

of ASP.NET and the underlying .NET Framework are not covered. If you are brand new to ASP.NET, be sure to also check out *Beginning ASP.NET 1.1 with VB .NET 2003* (ISBN: 0-7645-5707-6) or *Beginning ASP.NET 1.1 with Visual C# .NET 2003* (ISBN: 0-7645-5708-4), depending on your language of choice, to help you understand the basics of ASP.NET.

Is this book for the Visual Basic developer or for the C# developer? I am happy to say—BOTH! This book covers all examples in both VB and C# if the code differs considerably.

What This Book Covers

As I stated, this book spends its time reviewing the big changes that have occurred in the 2.0 release of ASP.NET. After the introduction, each major new feature included in ASP.NET 2.0 is covered in more detail. The following sections present the contents of each chapter.

Chapter 1: Introduction to ASP.NET 2.0

This first chapter gives a good grounding in the new features of ASP.NET 2.0. The chapter takes a look at some of the major new features and capabilities included. It starts by giving you a little bit of history of ASP.NET and, for those working with a beta for the first time, it explains what a beta build of a product is and what to expect from it.

Chapter 2: Visual Studio 2005

This chapter takes a look at the next generation of the major IDE for developing .NET applications: Visual Studio 2005. Previous releases of this IDE included Visual Studio .NET 2003 and Visual Studio .NET 2002. This chapter focuses on the enhancements in the 2005 release and how you can use it to build better ASP.NET applications more quickly than in the past.

Chapter 3: Application and Page Frameworks

The third chapter covers the frameworks of ASP.NET applications as well as the structure and frameworks provided for single ASP.NET pages. This chapter shows you how to build ASP.NET applications using IIS or the built-in Web server that now comes with Visual Studio 2005. This chapter also shows you the new folders and files added to ASP.NET. It also covers new ways of compiling code and how to perform cross-page posting.

Chapter 4: New Ways to Handle Data

ADO.NET incorporates some radical changes. This chapter takes a look at the new data model provided by ASP.NET, which allows you to handle the retrieval, updating, and deleting of data quickly and logically. This new data model enables you to use one or two lines of code to get at data stored in everything from SQL Server to XML files.

Chapter 5: Site Navigation

It is quite apparent that many developers do not simply develop single pages. Developers build *applications* and, therefore, they need mechanics that deal with functionality throughout the entire application, not just the pages. One of the new application capabilities provided by ASP.NET 2.0 is the site navigation system covered in this chapter. The underlying navigation system enables you to define your application's navigation structure through an XML file. Finally, it introduces a whole series of new navigation server controls that work with the data from these XML files.

Chapter 6: Working with Master Pages

In addition to the new site navigation system provided by ASP.NET 2.0—for working with the entire application as opposed to working with singular pages—the ASP.NET team developed a way to create templated pages. This chapter examines the creation of these templates (known as master pages) and how to apply them to your content pages throughout an ASP.NET application.

Chapter 7: Themes and Skins

CSS files provided in ASP.NET 1.0/1.1 are simply not adequate, especially in the area of server controls. The developer is never sure of the HTML output that is generated. This chapter takes a look at how to deal with the styles that your applications require. I look closely at how to create themes and the skin files that are part of a theme.

Chapter 8: Membership and Role Management

This chapter covers the new membership and role management system developed to simplify adding authentication and authorization to your ASP.NET applications. These two new systems are extensive and make some of the more complicated authentication and authorization implementations of the past a distant memory. The chapter focuses on using the `web.config` file for controlling how these systems are applied, as well as the new server controls that work with the underlying systems.

Chapter 9: Personalization

Developers are always looking for ways to store information pertinent to the end user. After it is stored, this personalization data has to be persisted for future visits or for grabbing other pages within the same application. The ASP.NET team developed a way to store this information—the ASP.NET personalization system. The great thing about this system, like the other systems introduced before it, is that you configure the entire behavior of the system from the `web.config` file.

Chapter 10: Portal Frameworks and Web Parts

This chapter looks at Web Parts—a new way of encapsulating pages into smaller and more manageable objects. The great thing with Web Parts is that they can be made of a larger Portal Framework, which then can enable end users to completely modify how the Web Parts are constructed on the page—including the appearance and the layout of the Web Parts on the page.

Chapter 11: SQL Cache Invalidation

This chapter discusses the biggest change to the caching capabilities in ASP.NET—SQL cache invalidation. This new caching capability allows you to invalidate cached items based on changes that occur in the database. This new process ensures a new way of keeping your pages as fresh as possible, but use the smallest number of resources to do so.

Chapter 12: Additional New Controls

ASP.NET 2.0 contains more than 40 new server controls. Many of the controls are covered in the other chapters of the book, but this chapter looks at the new server controls still unexplained. Included in this chapter are discussions of the BulletedList, HiddenField, FileUpload, MultiView, View, Wizard, DynamicImage, and ImageMap server controls.

Chapter 13: Changes to ASP.NET 1.0 Controls

In addition to the new server controls that come with ASP.NET 2.0, you will find considerable changes have been made to the server controls that we all know and love from ASP.NET 1.0. This chapter takes a look at the traditional server controls that have changed.

Chapter 14: Administration and Management

Besides making it easier for the developer to be more productive in building ASP.NET applications, the ASP.NET team also put considerable focus into making it easier to manage the application. In the past, using ASP.NET 1.0/1.1, you managed the ASP.NET applications by changing values in an XML configuration file. This chapter provides an overview of the new GUI tools that come with this latest release that enable you to easily and effectively manage Web applications.

Chapter 15: Visual Basic 8.0 and C# 2.0 Language Enhancements

In addition to major changes to ASP.NET, considerable change has occurred in Visual Basic 8.0 and C# 2.0. The changes to these two languages, the primary languages used for ASP.NET development, are discussed in this chapter.

Conventions

I have used a number of different styles of text and layout in the book to help differentiate among various types of information. Here are examples of the styles I use and an explanation of what they mean:

- ❑ *New words* that I'm defining are shown in italics.
- ❑ Keys that you press on the keyboard, like Ctrl and Enter, are shown in initial caps and spelled as they appear on the keyboard.

Code appears in a number of different ways. If I'm talking about a code word in paragraph text—for example, when discussing the `if...else` loop—the code word is shown in this font. If it's a block of code that you can type as a program and run, it's shown on separate lines, within a gray box, like this:

```
public static void Main()
{
    AFunc(1,2,"abc");
}
```

Sometimes you see code in a mixture of styles, like this:

```
// If we haven't reached the end, return true, otherwise
// set the position to invalid, and return false.
pos++;
if (pos < 4)
    return true;
else {
    pos = -1;
    return false;
}
```

The code with a white background represents code I've already presented and that you don't need to examine further. The code with the gray background is what I want you to focus on at this point.

I demonstrate the syntactical usage of methods, properties, and so on using the following format:

```
SqlDependency="database:table"
```

Here, the italicized parts indicate *placeholder text*: object references, variables, or parameter values to be inserted.

Most of the code examples throughout the book are presented as numbered listings with descriptive titles, like this:

Listing 1-3: Targeting WML devices in your ASP.NET pages

Each listing is numbered as 1-3, where the first number represents the chapter number, and the number following the hyphen represents the sequential number for where that listing falls within the chapter. Downloadable code from the Wrox Web site (www.wrox.com) also uses this numbering system, so you can easily locate the examples you are looking for.

All code is shown in both VB and C# if warranted. The exception is for code in which the only difference is, for example, the value given to the `Language` attribute in the `Page` directive. In such situations, I don't repeat the code for the C# version; so the code is shown only once, as in the following example:

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>DataSetDataSource</title>
</head>
```

```
<body>
  <form id="form1" runat="server">
    <asp:DropDownList ID="Dropdownlist1" Runat="server" DataTextField="name"
      DataSourceID="DataSetDataSource1">
    </asp:DropDownList>

    <asp:DataSetDataSource ID="DataSetDataSource1" Runat="server"
      DataFile="~/Painters.xml">
    </asp:DataSetDataSource>
  </form>
</body>
</html>
```

Source Code

As you work through the examples in this book, you may choose either to type all the code manually or use the source code files that accompany the book. All the source code used in this book is available for download at <http://www.wrox.com>. When you get to the site, simply locate the book's title (either by using the Search box or one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 0-7645-7286-5.

After you download the code, just decompress it with your favorite compression tool. Alternatively, you can go to the main Wrox code download page at <http://www.wrox.com/dynamic/books/download.aspx> to see the code available for this book and all other Wrox books. Remember, you can easily find the code you are looking for by referencing the listing number of the code example from the book, such as Listing 1-3. I use these listing numbers when naming the downloadable code files.

Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, such as a spelling mistake or faulty piece of code, we would be very grateful if you would tell us about it. By sending in errata, you may save another reader hours of frustration; at the same time, you are helping us provide even higher quality information.

To find the errata page for this book, go to <http://www.wrox.com> and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page, you can view all errata that have been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at <http://www.wrox.com/misc-pages/booklist.shtml>.

If you don't spot "your" error already on the Book Errata page, go to <http://www.wrox.com/contact/techsupport.shtml> and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

p2p.wrox.com

For author and peer discussion, join the P2P forums at p2p.wrox.com. The forums are a Web-based system for you to post messages relating to Wrox books and technologies and to interact with other readers and technology users. The forums offer a subscription feature that enables you to receive e-mail on topics of interest when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are represented in these forums.

At <http://p2p.wrox.com> you will find a number of different forums that will help you not only as you read this book, but also as you develop your own applications. To join the forums, just follow these steps:

1. Go to p2p.wrox.com and click the Register link.
2. Read the terms of use and click Agree.
3. Supply the information required to join, as well as any optional information you want to provide, and click Submit.

You will receive an e-mail with information describing how to verify your account and complete the joining process.

You can read messages in the forums without joining P2P, but you must join in order to post messages.

After you join, you can post new messages and respond to other users' posts. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how the forum software works, as well as answers to many common questions specific to P2P and Wrox books, be sure to read the P2P FAQs. Simply click the FAQ link on any P2P page.

The Challenge of Writing a Beta Book

This book, *ASP.NET 2.0 Beta Preview*, was my seventh book, and it was quite a different experience from other books I've written. I started writing this book at the end of 2003 when ASP.NET 2.0 was literally in pieces, and there wasn't an IDE to work with. Throughout the writing process, I received many new builds—some on CD, some on DVD, some for which I could log on to a remote server and work only remotely with the builds. With each new build I received, I uninstalled previous builds and installed the new one. Sometimes I had to completely rebuild my machine.

The look and feel of the IDE—Visual Studio 2005—changed considerably from one build to the next, as well as names of classes, methods, and properties. I say I wrote this book, but I probably wrote it at least two times during the process. I tell this story because it is important for you to understand that I wrote this book using numerous ASP.NET builds. In order to get this book to you in a timely fashion, the end product, which is in your hands, was retested and changed for the last build I received during the writing process. That particular build was one that was also received by more than 10,000 people at TechEd 2004 in San Diego, California—the May 2004 Technology Preview of Visual Studio 2005.

Introduction

Therefore, the code and the screen shots of the IDE you see throughout this book may be slightly different than the build (whether it is Beta 1 or Beta 2 of ASP.NET) you are working with. I don't expect the look and the feel of the IDE or the names of classes, methods, and properties to change again—but I can never be completely sure. Just be aware that you may encounter some slight differences between what you see on your screen and what I show in the book.

Acknowledgments

I have said it before and I'll say it again: Writing a book may seem like the greatest of solo endeavors, but it requires a large team of people working together to get technical books out the door—and this book is no exception. Wrox put together a top-notch team to bring information on this outstanding new technology to you as quickly as possible, as well as to ensure the highest quality content. I would first like to thank Jim Minatel, the senior acquisitions editor on the book. Thanks for letting me take on this wonderful project.

One of the biggest thanks goes to Jodi Jensen, the book's development editor. I have worked with Jodi on numerous books, and she is by far the best development editor out there. At the start of the project, I knew how tough it was going to be and wanted the best DE I know to work with me. Jodi is simply the best there is (I even tried to get a clause added to my contract stating that she would be the DE)!

Huge thanks go to Devin Rader for his work as the book's technical editor. Devin had to deal with the issue of the ever-changing builds, just as I did. Our conversations concerning chapters usually began "What build did you write that with?" Devin is an outstanding technical editor and has edited a bunch of my books before this one. I am indebted to him for his hard work on this project and all the projects before this. Devin, who is getting married (finally!) this fall, also works with me on INETA.

Peter Lanoie was gracious enough to donate his time to providing technical and usability feedback on several chapters, which was greatly appreciated.

Additional thanks go to Joe Wikert (Wrox publisher), Mary Lagu (copy editor extraordinaire), Jennifer Webb (marketing manager), and Brian Herrmann (development editor).

I would also like to thank Kent Sharkey, Rob Howard, Shawn Nandi, Scott Guthrie, and Brian Goldfarb—all from the ASP.NET team—for their help with all the questions I posed throughout the writing process. Thanks guys, I really do appreciate it!

I travel quite a bit for my job, and I wrote much of this manuscript during late nights in various hotels. I find it interesting to note that I started this project somewhere over the Atlantic Ocean on my way to London and finished writing it sitting on the third floor of the New York Public Library at 7 p.m. on June 10, 2004 (which is also my 10th wedding anniversary).

Unfortunately, writing takes time away from the family. I am lucky that I have the most loving and understanding wife in the world. Therefore, I thank my wife, Tuija, for putting up with my perpetual writing habit and helping me with all the loose ends that I lose track of but still need to tie up. My work would not be possible without her help and love. I also want to thank my kids—Henri Oskari and Sofia Amanda. Many times during this project, they would bustle in early on a Saturday morning and ask what I was doing. "Writing a book," I would answer. "Ohhhhh nooooo, not another book," they would wail. These kids kept me sane by convincing me that I had to step away from the desk and play games with them—something I wish I could do every day of the week!

1

Introduction to ASP.NET 2.0

The evolution of ASP.NET continues! The progression from Active Server Pages 3.0 to ASP.NET 1.0 was revolutionary, to say the least — and I am here to tell you that the evolution from ASP.NET 1.0/1.1 to ASP.NET 2.0 is just as exciting and dramatic.

The introduction of ASP.NET 1.0 changed the Web programming model, but ASP.NET 2.0 is just as revolutionary in the area of productivity. The primary goal of ASP.NET 2.0 is to enable you to build powerful, secure, and dynamic applications using the least possible amount of code. This book focuses on the astounding new capabilities that are built into ASP.NET 2.0.

In writing this book, I assume that you are already familiar with ASP.NET 1.0 or 1.1. I do not cover the basic functionality of ASP.NET provided by those releases.

This book focuses on the Beta 1 release of ASP.NET 2.0. A *beta* release is a software release that comes out prior to the final release of the product (the final release is often referred to as the *RTM* or *Release to Manufacturer* version). Software companies sometimes release products early as betas in hopes that the programming community will demand the features and capabilities that the release offers. The vendors also hope that the beta version will reveal any bugs in the product prior to the release of the RTM version. Therefore, be aware that you might encounter errors or bugs as you code your applications in the ASP.NET 2.0 beta release. Also be aware that the method or parameter names might change between the beta version and the RTM version. You may have to rework any ASP.NET applications built using the ASP.NET 2.0 beta when the RTM version is released. A beta, however, gives you an outstanding opportunity to gain early insight into the direction a new technology is going and to get up to speed on its use, even before it is released.

A Little Bit of History

ASP.NET 2.0 has its roots in an older Web technology from Microsoft, which was called Active Server Pages — or ASP for short. ASP was a quick and easy way to develop Web pages. ASP pages consisted of a single page that contained a mix of languages. The power of ASP was that you could use VBScript or JavaScript code instructions in the pages that would then be executed on the Web server prior to the page being sent to the end user's Web browser. This was an easy way to create dynamic Web pages that could be customized based on parameters dictated by the developer.

ASP 2.0 and 3.0 were quite popular because this technology made it relatively straightforward and easy to create Web pages. Also, ASP 2.0 and 3.0 appeared in the late '90s, just as the dotcom era was born. During this time, a mountain of new Web pages and portals were developed. ASP was one of the leading technologies that individuals and companies used to build them. In fact, even today, you can still find a lot of .asp pages on the Internet — including some of Microsoft's own Web pages.

But even at the time of the final release of Active Server Pages, in late 1998, Microsoft employees Marc Anders and Scott Guthrie had other ideas. Their ideas generated what they called XSP (which was an acronym with no meaning) — a new way of creating Web applications in an object-oriented manner instead of the interpreted manner of ASP 3.0. They showed their idea to many different groups within Microsoft, and it was quite well received. In the summer of 2000, the beta of what then was called ASP+ was released at Microsoft's Professional Developers Conference where the attendees eagerly started working with it. When the technology became available (with the final release of the .NET Framework 1.0), it was renamed ASP.NET — receiving the .NET moniker that most of Microsoft's new products were receiving at that time.

In the summer of 2000, and throughout the entire beta program for ASP+, this outstanding new technology created excitement. At this point, the entire .NET Framework was rather immature. The code for the entire Framework came on a single CD. No IDE came with it to enable development of ASP+ pages. To create your pages and code-behind classes, you had to use Microsoft's Notepad and the command-line compilers contained on the CD. I am happy to say that even today — in ASP.NET 2.0 — you can *still* use this simple approach to code your applications if you want!

Just working with the first ASP.NET beta was exciting; it is no different with the beta this time around. Nothing is better than getting your hands on a new technology and seeing what is *possible*. The following section discusses the goals of ASP.NET 2.0. See what you can expect from this new beta.

The Goals of ASP.NET 2.0

ASP.NET 2.0 is a major release of the product and is a built-in part of the .NET Framework 2.0. This release of the Framework is code-named *Whidbey*. You might hear others referring to this release of ASP.NET as *ASP.NET Whidbey*. ASP.NET 2.0 heralds a new wave of development that should eliminate any of the remaining barriers to adopting this new way of coding Web applications.

When the ASP.NET team started working on ASP.NET 2.0, it had specific goals to achieve. These goals focused around developer productivity, administration and management, performance and scalability, and the capability to target *any* device. They were completely achieved with this milestone product release. The next sections take a look at each of these goals.

Developer productivity

Much of the focus of ASP.NET 2.0 is on productivity. Huge productivity gains were made in going from ASP 3.0 to ASP.NET — could there possibly be much more left to gain?

One goal the development team had for ASP.NET 2.0 was to eliminate much of the intense coding that ASP.NET required and to make ASP.NET tasks easier. The ASP.NET team developing ASP.NET 2.0 had the goal of reducing by two-thirds the number of lines of code required for an ASP.NET application! It succeeded in this release, and you will find it literally amazing how quickly you can create an application in ASP.NET.

The new developer productivity capabilities are the focus of much of the book, so you can find examples on almost every page. But first, take a look at the older technology. In Listing 1-1, you use ASP.NET 1.0 to build a table in a Web page that includes simple paging of the data.

Listing 1-1: Showing data in a DataGrid server control with paging enabled (VB only)

```
<%@ Page Language="VB" AutoEventWireup="True" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<script runat="server">

    Private Sub Page_Load(ByVal sender As System.Object, _
        ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            BindData()
        End If
    End Sub

    Private Sub BindData()
        Dim conn As SqlConnection = New SqlConnection("server='localhost';
            trusted_connection=true; Database='Northwind'")
        Dim cmd As SqlCommand = New SqlCommand("Select * From Customers", conn)
        conn.Open()

        Dim da As SqlDataAdapter = New SqlDataAdapter(cmd)
        Dim ds As New DataSet

        da.Fill(ds, "Customers")

        DataGrid1.DataSource = ds
        DataGrid1.DataBind()
    End Sub

    Private Sub DataGrid1_PageIndexChanged(ByVal source As Object, _
        ByVal e As System.Web.UI.WebControls.DataGridPageChangedEventArgs)
        DataGrid1.CurrentPageIndex = e.NewPageIndex
        BindData()
    End Sub

</script>
<html>
```

(continued)

Listing 1-1: *(continued)*

```
<head>
</head>
<body>
    <form runat="server">
        <asp:DataGrid id="DataGrid1" runat="server" AllowPaging="True"
            OnPageIndexChanged="DataGrid1_PageIndexChanged"></asp:DataGrid>
    </form>
</body>
</html>
```

Although quite a bit of code is used here, this is a dramatic improvement over what was required to accomplish this task using Active Server Pages 3.0. I won't go into the details of the code; I just want to demonstrate that in order to add additional common functionality (such as paging) for the data shown in a table, the developer had to create custom code.

This is one area where the new developer productivity gains are most evident. ASP.NET 2.0 now provides a new control called the GridView server control. This control is much like the DataGrid server control that you may already know and love, but the GridView server control contains the built-in capability to apply paging, sorting, and editing of data with relatively little work on your part (besides offering many more new features). Look at an example of the GridView server control in Listing 1-2. This example builds a similar table of data from the Customers table in the Northwind database that now includes paging.

Listing 1-2: Viewing a paged dataset with the new GridView server control

```
<%@ page language="VB" %>

<script runat="server">

</script>

<head id="Head1" runat="server">
<head runat="server">
    <title>GridView Demo</title>
</head>
<body>
    <form runat="server">
        <asp:GridView ID="GridView1" Runat="server" AllowPaging="True"
            DataSourceId="SqlDataSource1" />
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
            SelectCommand="Select * From Customers"
            ProviderName="System.Data.OleDb"
            ConnectionString="Provider=SQLOLEDB;Server=localhost;uid=sa;
                pwd=password;database=Northwind" />
    </form>
</body>
</html>
```

That's it! You can apply paging by using a couple of new server controls. You turn on this capability using a server control attribute, the `AllowPaging` attribute of the `GridView` control:

```
<asp:GridView ID="GridView1" Runat="server" AllowPaging="True"
    DataSourceId="SqlDataSource1" />
```

The other interesting event occurs in the code section of the document:

```
<script runat="server">

</script>
```

These two lines of code aren't actually needed to run the file, but I include them here to make a point — *you don't need to write any server-side code to make this all work!* You only have to include some server controls: one control to get the data and one control to display the data. The controls are then wired together. Running this page produces the results shown in Figure 1-1.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
GOURL	Gourmet Lanchonetes	André Fonseca	Sales Associate	Av. Brasil, 442	Campanas	SP	04876-786	Brazil	(11) 555-9482	
GREAL	Great Lakes Food Market	Howard Snyder	Marketing Manager	2732 Baker Blvd	Eugene	OR	97403	USA	(503) 555-7555	
GROSR	GROSELLA-Restaurante	Manuel Pereira	Owner	5ª Ave. Los Palos Grandes	Caracas	DF	1081	Venezuela	(2) 283-2951	(2) 283-3397
HANAR	Hanari Carnes	Mario Pontes	Accounting Manager	Rua do Paço, 67	Rio de Janeiro	RJ	05454-876	Brazil	(21) 555-0091	(21) 555-8765
HILAA	HILARION-Abastos	Carlos Hernández	Sales Representative	Carrera 22 con Ave. Carlos Soublette #8-35	San Cristóbal	Táchira	5022	Venezuela	(5) 555-1340	(5) 555-1948
HUNGC	Hungry Coyote Import Store	Yoshi Latimer	Sales Representative	City Center Plaza 516 Main St	Elgin	OR	97827	USA	(503) 555-6874	(503) 555-2376
HUNGO	Hungry Owl All-Night Grocers	Patricia McKenna	Sales Associate	8 Johnstown Road	Cork	Co. Cork		Ireland	2967 542	2967 3333
ISLAT	Island Trading	Helen Bennett	Marketing Manager	Garden House Crowther Way	Cowes	Isle of Wight	PO31 7PJ	UK	(198) 555-8888	
KOENE	Königlich Essen	Philip Cramer	Sales Associate	Maubelstr. 90	Brandenburg		14776	Germany	0555-09876	
LACOR	La corne d'abondance	Daniel Tonini	Sales Representative	67, avenue de l'Europe	Versailles		78000	France	30.59.84.10	30.59.85.11

1 2 3 4 5 6 7 8 9 10

Figure 1-1

This is just one of thousands of possible examples, so at this point you can't possibly grasp how much more productive you can be with ASP.NET 2.0. As you work through the book, however, you will see plenty of examples that demonstrate this new level of productivity.

Administration and management

The initial release of ASP.NET focused on the developer, and little was geared toward the people who had to administer and manage all the ASP.NET applications that were built. Instead of working with consoles and wizards as they did in the past, administrators and managers of these new applications now had to work with XML configuration files such as `machine.config` and `web.config`.

To remedy this situation, ASP.NET 2.0 now includes a Microsoft Management Console (MMC) snap-in that enables Web application administrators to easily edit configuration settings on the fly. Figure 1-2 shows the ASP.NET Configuration Settings dialog open on one of the available tabs.

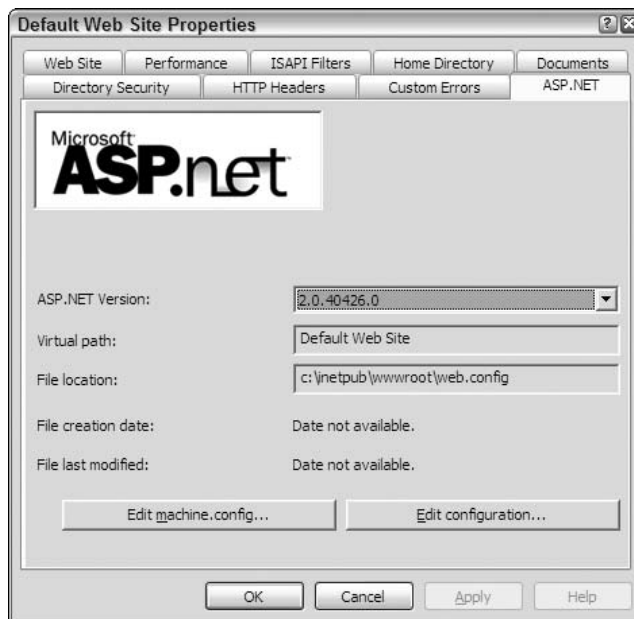


Figure 1-2

This dialog allows system administrators to edit the contents of the `machine.config` and the `web.config` files directly from the dialog instead of having to examine the contents of an XML file.

In addition to this dialog, Web or system administrators have another way to administer their ASP.NET 2.0 applications — using the new Web Administration Tool shown in Figure 1-3.

You might be asking yourself how you can access these new tools. Well, that is the exciting part. These tools are built off new APIs that are now part of the .NET Framework 2.0 and which are open to you as a developer. These new APIs give you programmatic access to many of the configurations of your Web applications. You now have programmatic access to reading and writing to `.config` files, enabling you to create similar tools or even deployment and management scripts.

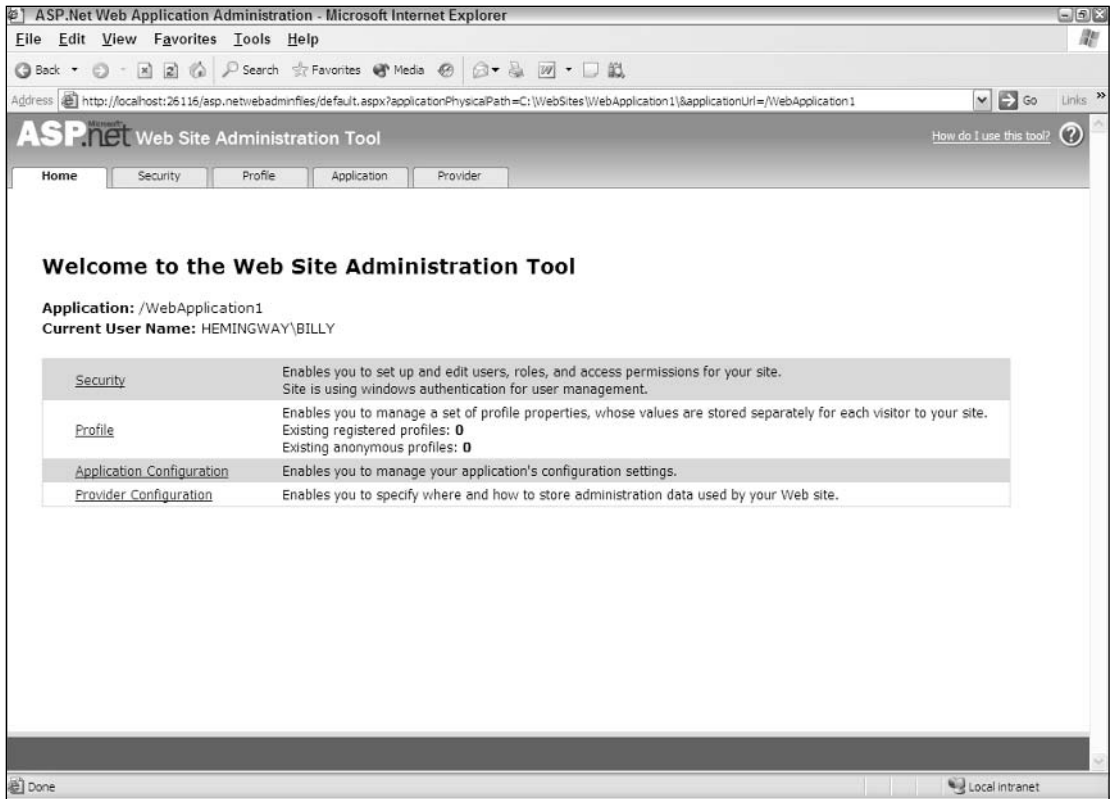


Figure 1-3

In addition to these new capabilities, you now also have the capability to easily encrypt sections of your configuration files. In the past, many programmers stored vital details — such as usernames, passwords, or even their SQL connection strings — directly in the `web.config` file. With the capability to easily encrypt sections of these files, you can now store these items in a more secure manner.

Performance and scalability

One of the goals for ASP.NET 2.0 was to provide the world's fastest Web application server. ASP.NET 2.0 includes a number of performance enhancements that are addressed throughout this book.

One of the most exciting performance enhancements is the new caching capability aimed at Microsoft's SQL Server. ASP.NET 2.0 now includes a feature called *SQL cache invalidation*. Before ASP.NET 2.0, it was possible to cache the results that came from SQL Server and to update the cache based upon a time interval — for example, every 15 seconds or so. This meant that the end user might see stale data if the result set changed sometime during that 15-second period.

In some cases, this time interval result set is unacceptable. In an ideal situation, the result set stored in the cache is destroyed if any underlying change occurred in the source from which the result set was retrieved — in this case, SQL Server. With ASP.NET 2.0, you can make this happen with the use of SQL cache invalidation. This means that when the result set from SQL Server changes, the output cache is triggered to change, and the end user always sees the latest result set. The data presented is never stale.

Chapter 1

Another big area of change in ASP.NET is in the area of performance and scalability. ASP.NET 2.0 now provides 64-bit support. This means that you can now run your ASP.NET applications on 64-bit Intel or AMD processors.

Because ASP.NET 2.0 is fully backward compatible with ASP.NET 1.0 and 1.1, you can now take any former ASP.NET application, recompile the application on the .NET Framework 2.0, and run it on a 64-bit processor.

Device-specific code generation

If you thought that building device applications with ASP.NET 1.0 or 1.1 was easy in the past, wait until you see how you accomplish this in ASP.NET 2.0. ASP.NET 1.0 gave you the capability to build mobile applications through the additional download of the Microsoft Mobile Internet Toolkit (MMIT). With ASP.NET 1.1, this was included by default and, therefore, didn't require the download. It did, however, still require the use of `<mobile:>` server controls instead of the standard `<asp:>` server controls that ASP.NET provided.

With ASP.NET 2.0, you no longer use the `<mobile:>` server controls. All the `<asp:>` server controls now have the capability to output to various devices and not just the *big* browsers, such as Microsoft Internet Explorer or Opera. The `<asp:>` server controls can now output not only in HTML, but also in XHTML, CHTML, or WML (see Figure 1-4).

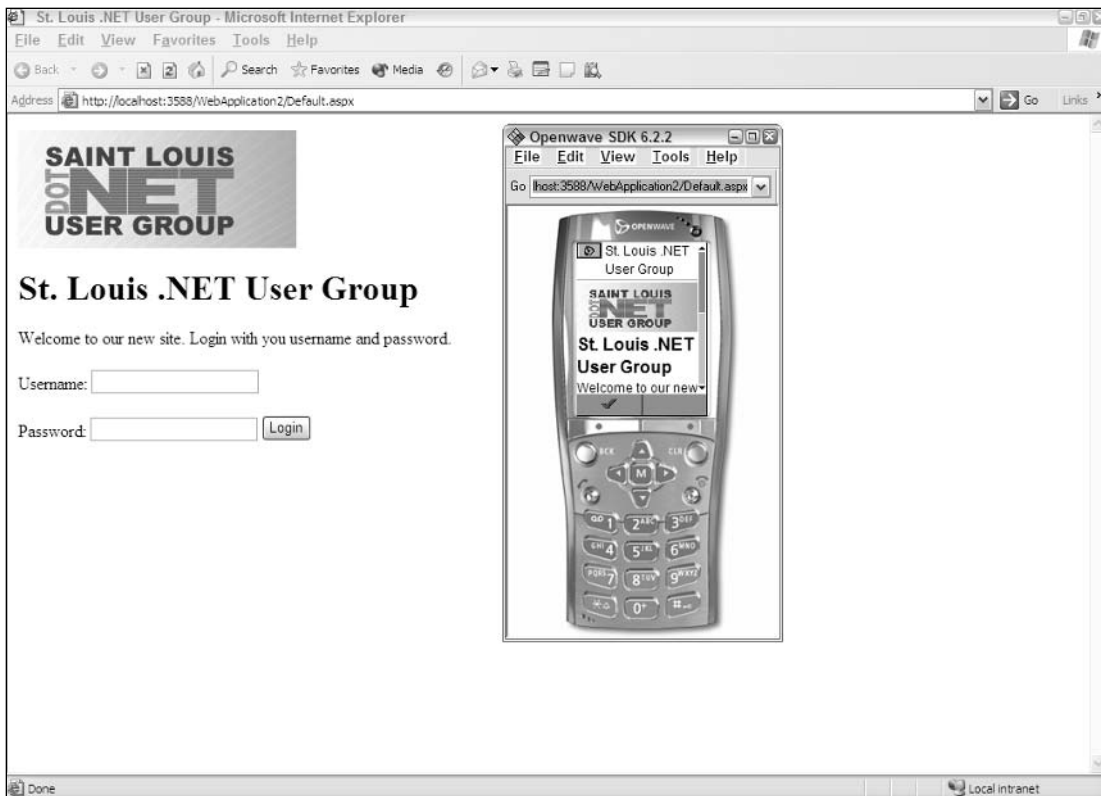


Figure 1-4

ASP.NET 2.0 determines the output based on the target, or you can apply any number of filters to fine-tune the output based on the device targeting your pages. For example, to have ASP.NET produce specific output for any WML device that calls the page, you do something similar to what is shown in Listing 1-3.

Listing 1-3: Targeting WML devices in your ASP.NET pages

```
<asp:DynamicImage ID="DynamicImage1" Runat="server" ImageFile="header-1.gif"
  wml:ImageFile="header-2.gif">
</asp:DynamicImage>
```

In the end, by the use of ASP.NET 2.0, you can better target devices that come to your site or application without building a separate portal.

Additional New Features of ASP.NET 2.0

You just learned some of the main goals of the ASP.NET team who built ASP.NET 2.0. To achieve these goals, it built a mountain of new features into ASP.NET. I describe a few of them here.

New developer infrastructures

An exciting advancement in ASP.NET 2.0 is that new infrastructures are in place for you to use in your applications. The ASP.NET team determined some of the most common programming operations that users were performing with ASP.NET 1.0 and decided to build a few of these operations directly into ASP.NET itself. The fact that these infrastructures are now built directly into the .NET Framework saves you considerable time and coding.

Membership and role management

In earlier versions, if you were developing a portal that required users to log in to the application to gain privileged access, invariably you had to create it yourself. It can be tricky to create applications with areas that are accessible only to select individuals.

With ASP.NET 2.0, this capability is now built in. You can now validate users as shown in Listing 1-4.

Listing 1-4: Validating a user in code**VB**

```
If (Membership.ValidateUser (Username.Text, Password.Text)) Then
    ' Allow access code here
End If
```

C#

```
If (Membership.ValidateUser (Username.Text, Password.Text)) {
    // Allow access code here
}
```

Chapter 1

A new series of APIs in ASP.NET 2.0 enable you to control an application's user membership and role management. Using these APIs, you can easily manage users and their complex roles — creating, deleting, and editing them. You get all this capability by using the APIs or a built-in Web tool called the Web Site Administration Tool.

As far as storing users and their roles, ASP.NET 2.0 is geared to work with Microsoft Access, SQL Server, or Active Directory out of the box. By default, ASP.NET uses an .mdb file (Access) for storing all users and roles. You are in no way limited to just one of these three data stores, however. You can expand everything offered to you by ASP.NET and build your own providers using whatever you fancy as a data store. For example, if you want to build your user store in LDAP or within an Oracle database, you can do so quite easily.

Personalization

One advanced feature that portals love to offer their membership base is the capability to personalize their offerings so that end users can make the site look and function however they want. The capability to personalize an application and store the personalization settings is now completely built into the ASP.NET framework.

Because personalization usually revolves around a user and possibly a role that this user participates in, the personalization architecture can be closely tied to the membership and role infrastructures. You have a couple of options as to where you can store the created personalization settings. The capability to store these settings in either Microsoft Access or in SQL Server is built into ASP.NET 2.0. As with the capabilities of the membership and role APIs, you can use the flexible provider model that is offered, and then either change how the built-in provider uses the available data store or build your own custom data provider to work with a completely new data store. The personalization API also supports a union of data stores, meaning that you can use more than one data store if you want.

Because it is so easy to create a site for customization using these new APIs, this feature is quite a value-add for any application you build.

The ASP.NET Portal Framework

During the days of ASP.NET 1.0, developers could go to the ASP.NET team's site (found at <http://www.asp.net>) and download some Web application demos called IBuySpy. Known as Developer Solution Kits, these demos were used as the basis for many of the Web sites on the Internet today.

The nice thing about IBuySpy was that you could use the code that it provided as a basis to build either a Web store or a portal. You simply took the base code and extended it. For example, you could change the look and feel of the presentation part of the code or introduce advanced functionality into its modular architecture. Developer Solution Kits were quite popular because they made performing these types of operations so easy. Figure 1-5 shows the INETA (International .NET Association) Web site, which is built on the IBuySpy framework.

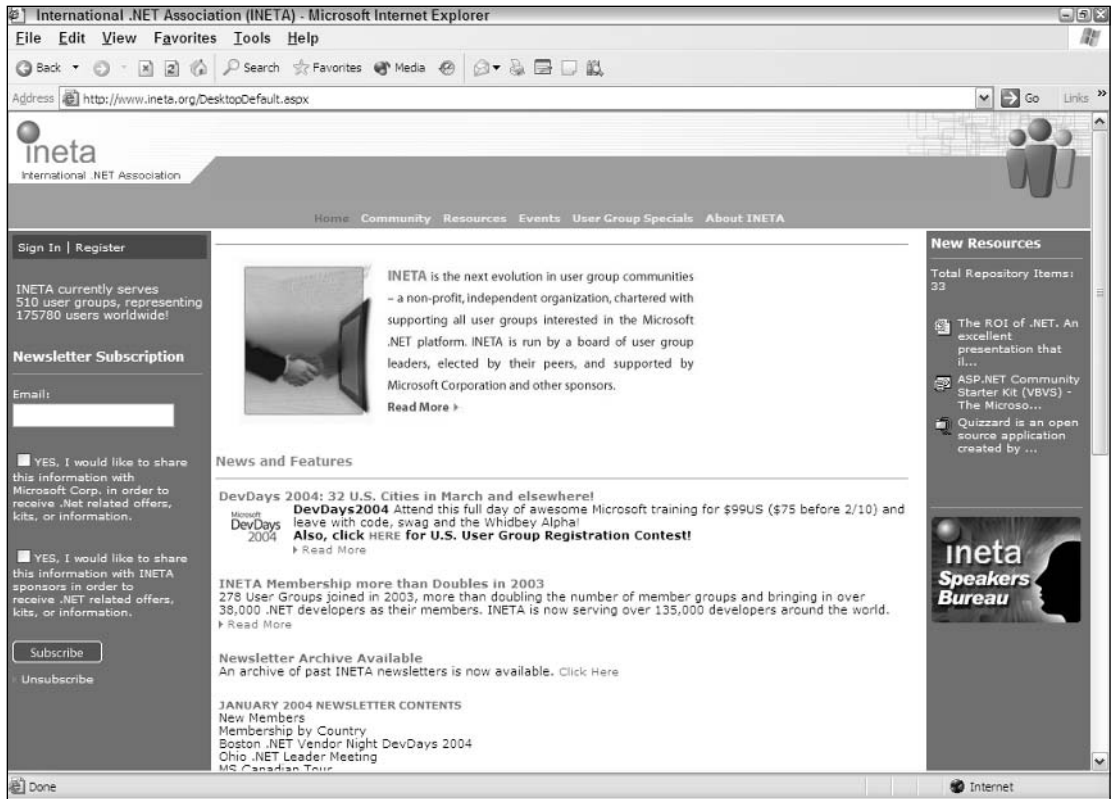


Figure 1-5

Because of the popularity of frameworks such as IBuySpy, ASP.NET 2.0 offers built-in capability for using Web Parts to easily build portals. The possibilities of what you can build using the new Portal Framework is astounding. The power of building using Web Parts is that it easily enables end-users to completely customize the portal for their own preferences. Figure 1-6 shows an example application built using Web Parts.

Site navigation

The ASP.NET team members realize that end users want to navigate through applications with ease. The mechanics to make this work in a logical manner is sometimes hard to code. The team solved the problem in ASP.NET 2.0 with a series of navigation-based server controls.

First, you can build a site map for your application in an XML file that specific controls can inherently work from. Listing 1-5 shows a sample site map file.



Figure 1-6

Listing 1-5: An example of a site map file

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap>
  <siteMapNode title="Home" description="Home Page" url="default.aspx">
    <siteMapNode title="News" description="The Latest News" url="News.aspx">
      <siteMapNode title="U.S." description="U.S. News"
        url="News.aspx?cat=us" />
      <siteMapNode title="World" description="World News"
        url="News.aspx?cat=world" />
      <siteMapNode title="Technology" description="Technology News"
        url="News.aspx?cat=tech" />
      <siteMapNode title="Sports" description="Sports News"
        url="News.aspx?cat=sport" />
    </siteMapNode>
    <siteMapNode title="Finance" description="The Latest Financial Information"
      url="Finance.aspx">
      <siteMapNode title="Quotes" description="Get the Latest Quotes"
        url="Quotes.aspx" />
      <siteMapNode title="Markets" description="The Latest Market Information"
```

```

url="Markets.aspx">
  <siteMapNode title="U.S. Market Report"
    description="Looking at the U.S. Market" url="MarketsUS.aspx" />
  <siteMapNode title="NYSE"
    description="The New York Stock Exchange" url="NYSE.aspx" />
</siteMapNode>
<siteMapNode title="Funds" description="Mutual Funds"
url="Funds.aspx" />
</siteMapNode>
<siteMapNode title="Weather" description="The Latest Weather"
url="Weather.aspx" />
</siteMapNode>
</siteMap>

```

After you have a site map in place, you can use this file as the data source behind a couple of new site navigation server controls, such as the TreeView and the SiteMapPath server controls. The TreeView server control enables you to place an expandable site navigation system in your application. Figure 1-7 shows you an example of one of the many looks you can give the TreeView server control.



Figure 1-7

The SiteMapPath is a control that provides the capability to place what some call navigation breadcrumbs in your application so that the end user can see the path that he has taken in the application and can easily navigate to higher levels in the tree. Figure 1-8 shows you an example of the SiteMapPath server control at work.

[Home](#) > [Finance](#) > [Markets](#) > U.S. Market Report

Figure 1-8

These new site navigation capabilities provide a great way to get programmatic access to the site layout and even to take into account things like end-user roles to determine which parts of the site to show.

Image generation

The DynamicImage server control is another exciting new control. Similar to the Image server control in ASP.NET 1.0, the DynamicImage control changes the image type based upon the container (browser or device type) of the end user. Not only does ASP.NET change its page output (HTML or WML, and so on) based upon the requester, but it also changes the file types of the images that are embedded in page output.

Figure 1-9 shows an image generated for a browser requesting the page. The file output is the same type as the file saved in the file system — a .gif file.

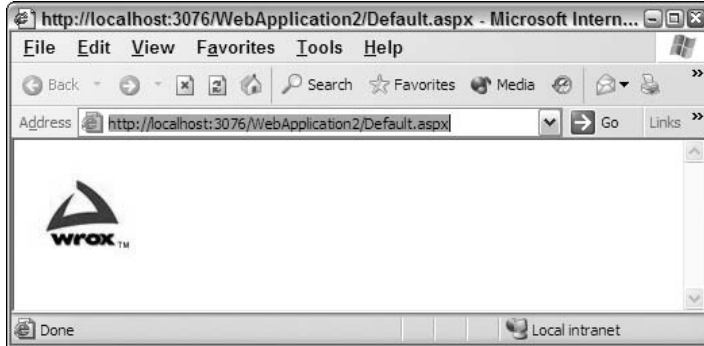


Figure 1-9

If a mobile device requests the same page, however, the file type is changed upon the request, and the output file is now suitable for the receiving device (a .wbmp file). This is shown in Figure 1-10.



Figure 1-10

New compilation system

In ASP.NET 2.0, the code is constructed and compiled in a new way. Compilation in ASP.NET 1.0 was always a tricky scenario. With ASP.NET 1.0, you could build an application using ASP.NET, deploy it, and then watch as it was compiled page by page as each page was requested. If you made any changes to the code-behind file in ASP.NET 1.0, it was not reflected in your application until the entire application was rebuilt. That meant that the same page-by-page request had to be done again before the entire application was recompiled.

Everything about how ASP.NET 1.0 worked with classes and compilation is changed with the release of ASP.NET 2.0. The mechanics of the new compilation system actually begin with how a page is structured in ASP.NET 2.0. In ASP.NET 1.0, you either constructed your pages using the code-behind model or by placing all the server code inline between `<script>` tags on your `.aspx` page. Most pages were constructed using the code-behind model because this was the default when using Visual Studio .NET 2002 or 2003. It was quite difficult to create your page using the inline style in these IDEs. If you did, you were then deprived of the use of IntelliSense, which can be quite the lifesaver when working with the tremendously large collection of classes that the .NET Framework offers.

ASP.NET 2.0 offers a new code-behind model mostly because the .NET Framework now offers the capability to work with partial classes (also called partial types). Upon compilation, the separate class files are combined into a single offering. This gives you much cleaner code-behind pages. The code that was part of the Web Form Designer Generated section of your classes is now separated from the code-behind classes that you create yourself.

ASP.NET 2.0 applications can include a `\Code` directory where you place your classes. Any class placed here is dynamically compiled and reflected in the application. You do not use a separate build process when you make changes as you did with ASP.NET 1.0. This is a “just save and hit” deployment model like the one in ASP 3.0. Visual Studio Web Developer also automatically provides IntelliSense for any objects that are placed in the `\Code` directory, whether you are working with the code-behind model or are coding inline.

ASP.NET 2.0 also provides you with tools that enable you to precompile your ASP.NET applications so that no page within your application has latency when it is retrieved for the first time. It is also a great way to figure out if you have made any errors in the pages without invoking every page yourself.

Precompiling your ASP.NET 2.0 applications is as simple as calling the `precompile.axd` imaginary file in the application root of your application after it has been deployed. This one call causes your entire application to be precompiled. You receive an error notification if any errors are found anywhere within your application. It is also possible to precompile your application and only deliver the created assembly to the deployment server, thereby protecting your code. You see examples of both of these scenarios later in this book.

Additions to the page framework

The ASP.NET page framework has some dramatic new additions that you can include in your applications. One of the most dramatic ones is the capability to build ASP.NET pages based upon visual inheritance. This was possible in the Windows Forms world, but it was harder to achieve with ASP.NET. You also gain the capability to easily apply a consistent look and feel to the pages of your application by using themes. Many of the difficulties in working with ADO.NET in the past have now been removed

Chapter 1

with the addition of a new series of data source controls that take care of accessing and retrieving data from a large collection of data stores. Although these are not the only new controls, the great number of new server controls create a larger ASP.NET page framework.

Master pages

With the introduction of *master pages* in ASP.NET 2.0, you can now use visual inheritance within your ASP.NET applications. Because many ASP.NET applications have a similar structure throughout their pages, it is logical to build a page template once and use that same template throughout the application.

In ASP.NET 2.0, you do this by creating a .master page, as shown in Figure 1-11.

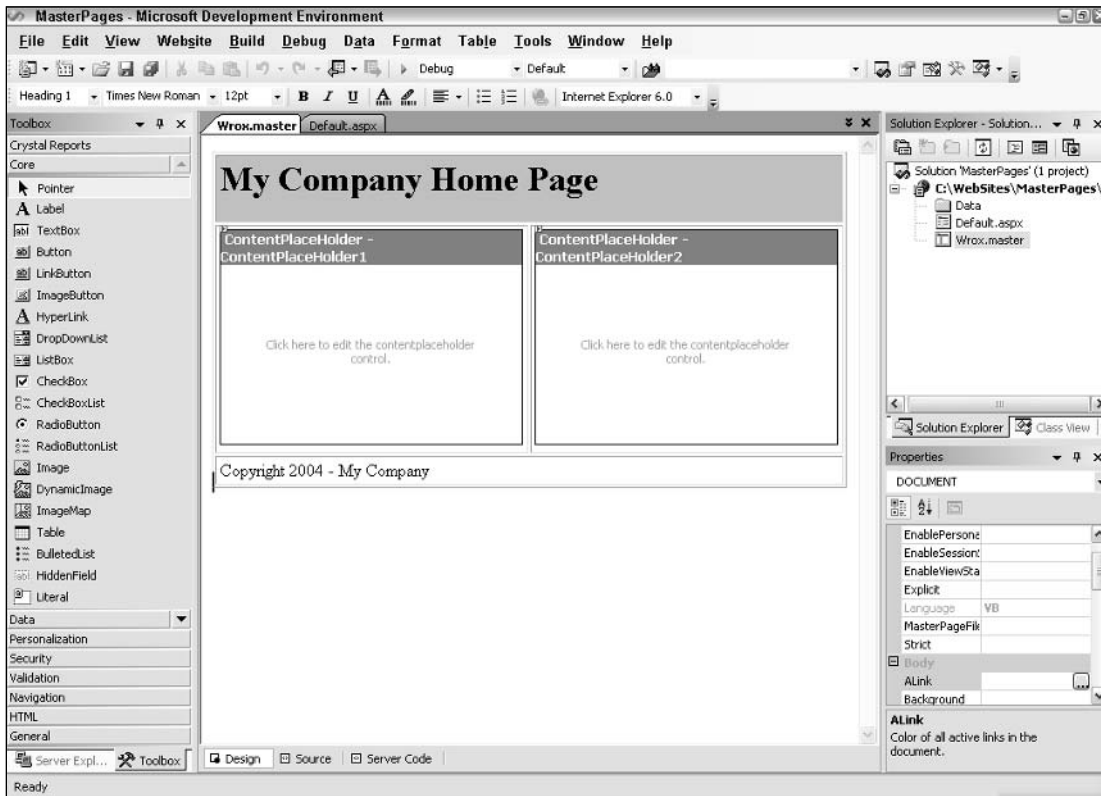


Figure 1-11

An example master page might include a header, footer, and any other elements that all the pages will share. Besides these core elements, which you might want on every page that inherits and uses this template, you can place `<asp:ContentPlaceHolder>` server controls within the master page itself for the subpages to use in order to change specific regions of the master page template. The editing of the subpage is shown in Figure 1-12.

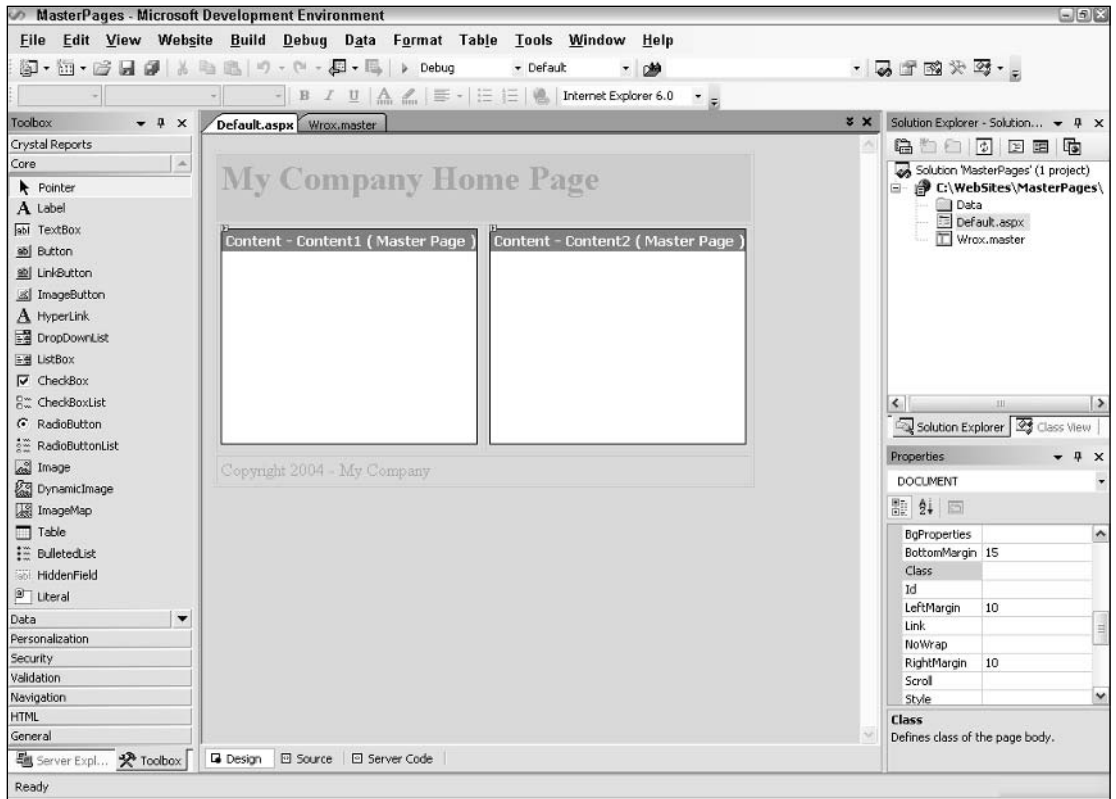


Figure 1-12

When an end user invokes one of the subpages, he is actually looking at a single page compiled from both the subpage and the master page that the particular subpage inherited from. This also means that the server and client code from both pages are enabled on the new single page.

The nice thing about master pages is that you now have a single place to make any changes that affect the entire site. This eliminates making changes to each and every page within an application.

Themes

The introduction of themes in ASP.NET 2.0 has made it quite simple to provide a consistent look and feel across your entire site. Themes are simple text files where you define the appearance of server controls that can be applied across the site, to a single page, or to a specific server control. You can also easily incorporate graphics and Cascading Style Sheets, in addition to server control definitions.

Themes are stored either on the server for all applications to use or in the /Theme directory within the application root for use within that particular application. The server-wide, pre-installed themes can be found at: C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\ASP.NETClientFiles\Themes.

One cool capability of themes is that you can dynamically apply them based on settings that use the new personalization service provided by ASP.NET 2.0. Each unique user of your portal or application can have her own personalized look and feel that she has set from your offerings.

New objects for accessing data

One of the more code-intensive tasks in ASP.NET 1.0 was in the retrieval of data. In many cases, this meant working with a number of objects. If you have been working with ASP.NET for a while, you know that it was an involved process to display data from a Microsoft SQL Server table within a DataGrid server control. For instance, you first had to create a number of new objects. They included a `SqlConnection` object followed by a `SqlCommand` object. When those objects were in place, you then created a `SqlDataReader` to populate your DataGrid by binding the result to the DataGrid. In the end, a table appeared containing the contents of the data you were retrieving (such as the Customers table from the Northwind database).

ASP.NET 2.0 eliminates this intensive procedure with the introduction of a new set of objects that work specifically with data access and retrieval. These new data controls are so easy to use that you access and retrieve data to populate your ASP.NET server controls without writing any code. You saw an example of this in Listing 1-2, where an `<asp:SqlDataSource>` server control retrieved rows of data from the Customers table in the Northwind database from SQL Server. This `SqlDataSource` server control was then bound to the new GridView server control via the use of simple attributes within the GridView control itself. It really couldn't be any easier!

The great news about this new functionality is that it is not limited to just Microsoft's SQL Server. In fact, a good number of data source server controls are at your disposal. You also have the capability to create your own. In addition to the `SqlDataSource` server control, ASP.NET 2.0 introduces `AccessDataSource`, `XmlDataSource`, `ObjectDataSource`, `DataSetDataSource`, and `SiteMapDataSource` server controls. You use all these new data controls later in this book.

New server controls

So far, you have seen a number of new server controls that you can use when building your ASP.NET 2.0 pages. For instance, I just spoke of all the new data source server controls that you can use to access different kinds of data stores. You also saw the use of the new GridView server control, which is an enhanced version of the previous DataGrid control that you used in ASP.NET 1.0.

Besides the controls presented thus far in this chapter, ASP.NET 2.0 provides more than 40 additional new server controls! In fact, so many new server controls have been introduced that the next IDE for building ASP.NET applications, Visual Studio 2005, had to reorganize the Toolbox where all the server controls are stored. They are now separated into categories instead of being displayed in a straight listing as they were in Visual Studio .NET or the ASP.NET Web Matrix. The new Visual Studio 2005 Toolbox is shown in Figure 1-13.



Figure 1-13

A New IDE for Building ASP.NET 2.0 Pages

With ASP.NET 1.0/1.1, you can build your ASP.NET application using Notepad, Visual Studio .NET 2002 and 2003, as well as the ASP.NET Web Matrix. ASP.NET 2.0 also introduces another IDE to the Visual Studio family — Visual Studio 2005.

Visual Studio 2005 offers some dramatic enhancements that completely change the way in which you build your ASP.NET applications. Figure 1-14 shows you a screen shot of the new Visual Studio 2005.

The most exciting change to the IDE is that Visual Studio 2005 builds applications using a file-based system, not the project-based system used by Visual Studio .NET. When using Visual Studio .NET, you had to create new projects (for example, an ASP.NET Web Application project). This process created a number of project files in your application. Because everything was based on a singular project, it became very difficult to develop applications in a team environment.

Chapter 1

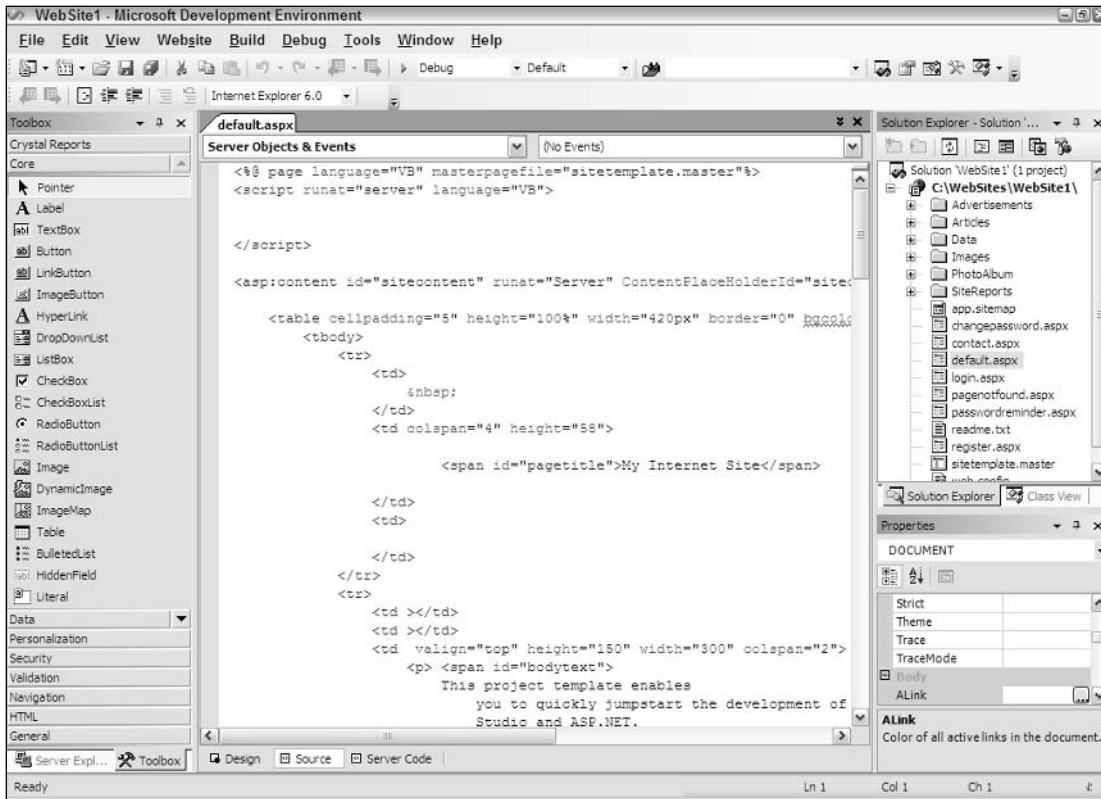


Figure 1-14

Visual Studio 2005, on the other hand, is based on a file system approach. No project files are included in your project, and this makes it very easy for multiple developers to work on a single application together without bumping into each other. Other changes are those to the compilation system, which I discussed earlier. You can now build your ASP.NET pages using the inline model or the new code-behind model. Whether you build pages inline or with the new code-behind model, you have full IntelliSense capabilities no matter what model you use. This, in itself, is powerful and innovative. Figure 1-15 shows IntelliSense running from an ASP.NET page being built using the inline model.

Another feature of Visual Studio 2005 that has come over from the ASP.NET Web Matrix is that you don't need ISS on your development machine. Visual Studio 2005 has a built-in Web server that enables you to launch pages from any folder in your system with relative ease. Chapter 2 discusses the new Visual Studio 2005 in detail.

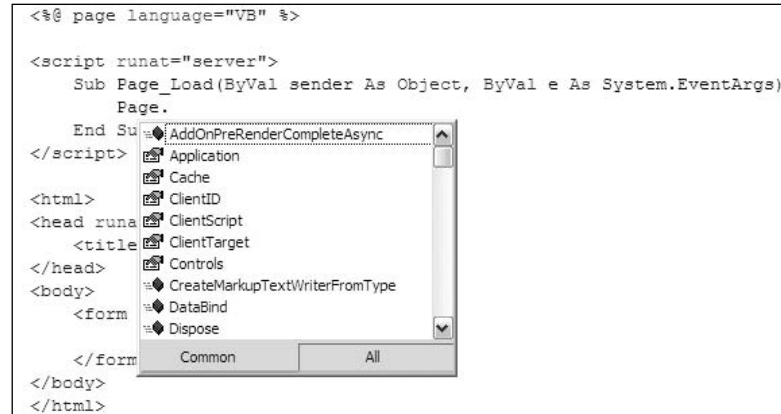


Figure 1-15

Summary

This chapter briefly introduced what is new in ASP.NET 2.0. This release offers so much that I didn't even come close to covering it all in this chapter. You will find the new ways of working with data, presentation, and the new infrastructures effective means to create powerful and secure applications. Even though ASP.NET 2.0 is presently only in a beta format, this is a great time to learn where the product is heading and how to use it.

ASP.NET 2.0 is so powerful and has so much capability built-in that its tremendous benefits to productivity really shine through.

2

Visual Studio 2005

Using the ASP.NET 2.0 beta requires that you also work with the beta of Visual Studio 2005 — the latest IDE from Microsoft to facilitate building .NET components and applications. Visual Studio 2005, building on Visual Studio .NET 2003, provides one of the best development environments for coding your ASP.NET applications.

Visual Studio 2005 enables you to build any type of .NET component or application. When you use this tool, you can choose any of the Microsoft .NET-compliant languages for building your applications, plus it allows you to create Windows Forms, XML Web services, .NET components, mobile applications, ASP.NET applications, and more. Included in this round are a large number of new wizards and *smart tags* that simplify the development process for you.

When you pull up Visual Studio 2005 for the first time on your computer, you select the environment in which you wish the IDE to open. This chapter assumes that you have selected *Web* because that environment is the focus of this book.

The next sections provides a quick tour of this new IDE.

The Document Window

The Document Window is where you create your ASP.NET pages. This section of the IDE enables you to create ASP.NET pages either by dragging and dropping elements onto a design surface or by directly coding them yourself.

Views in the Document Window

Visual Studio .NET 2002 and 2003 both had a Design view and an HTML view of the ASP.NET page. Visual Studio 2005 offers two views of a page: Design and Source. Figure 2-1 shows the Document Window in Visual Studio 2005.



Figure 2-1

The Document Window contains two tabs at the bottom that enable you to switch the view of your page: Design and Source. The Design tab enables you to view your ASP.NET page as it would appear in the browser. You use Design view to create your ASP.NET page visually in a WYSIWYG fashion. Dragging and dropping controls onto the design surface causes Visual Studio to generate code in the page. This is not very different from older versions of Visual Studio. The Source tab shows the complete source of the file and is the default view used by Visual Studio 2005.

You can change the default view that Visual Studio uses when a page is opened for the first time from the Options dialog. Choose Tools ⇨ Options and then navigate to the General section of the HTML Designer section. Here, you see the option to open pages in either the Design or Source view. Select the view you want and click OK. If you really dislike the Design view, you can actually lock it out by checking the Lock Design View check box found in the same location.

Although the Document Window is basically the same as it has always been, this section of the IDE does have some new functionality, which I describe in the following sections.

The tag navigator

When working visually with an ASP.NET page, notice that a list of the elements appears on your page at the bottom of the Document Window. This list of elements is called the *tag navigator* and is illustrated in Figure 2-2.

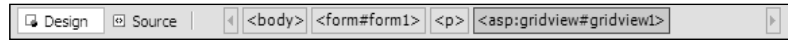


Figure 2-2

Instead of requiring you to select the element from the design surface or from within Code view, the tag navigator enables you to right-click on an element to select it and display the properties for that control in the Properties Window (discussed shortly). You can also select the content of the element by using this approach (see Figure 2-3).



Figure 2-3

When you have many elements on your page, the addition of the tag navigator is quite helpful. To use its capabilities, simply place your cursor in the Document Window and use the arrow buttons associated with the display to quickly scroll through elements to find what you are looking for. The tag navigator will show all the controls from the one you selected as well as all the selected control's child controls. When working in the Code view, you can, using the same mechanics, jump quickly to the content of the control. This new functionality is a quick and powerful way of navigating your page. You can also use this new functionality to highlight specific sections of code. For instance, to highlight everything inside a table, select the `<asp:Table>` element from the tag navigator, right-click the option, and select the content of the control. All the code between the opening `<asp:Table>` and the closing `</asp:Table>` elements is highlighted.

Page tabs

Another new and interesting feature of the Document Window is in how the page tabs work. Whenever you have a page open in the Document Window, a tab for that page appears at the top of the Document Window. When you have multiple documents open, this tabbed view of the pages enables you to switch quickly from one page to another simply by clicking the tab of the page you want to view. Although page tabs are not new to the IDE, the functionality that these tabs provide is certainly new. The following paragraphs explain this new functionality.

Right-clicking the page tab gives you the new options illustrated in Figure 2-4.

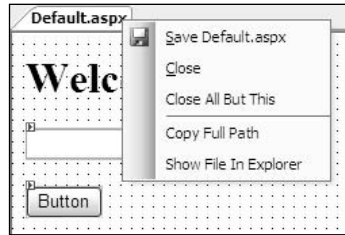


Figure 2-4

By right-clicking the page tab, you can save the file, close the file, close every open document but the one selected, display the full path of the file (such as `C:\WebSites\myWebApplication\Default.aspx`), and open the file in Windows Explorer (shown in Figure 2-5).

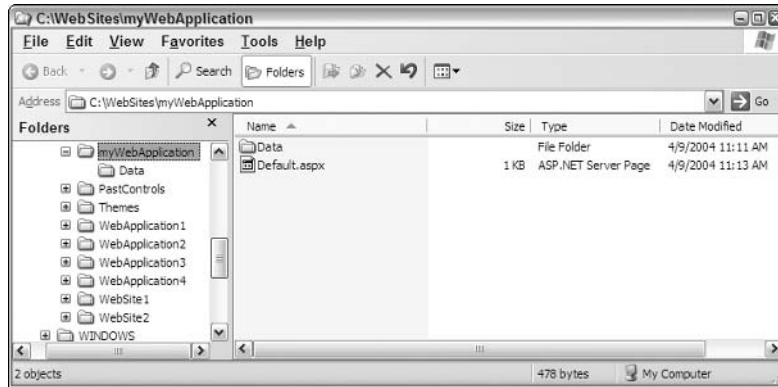


Figure 2-5

Code change status notifications

Some other changes to the Document Window include a new code-change notification system. When you work with code on your pages, notice that line numbers are now included by default. Clicking any number highlights that line of code. Next to the line numbers is a changing color bar, illustrated in Figure 2-6.

This color bar notifies you of code changes that have occurred on your ASP.NET pages. If no color bar appears on a particular line of code, you have not yet made any changes to that particular line. After you make a change to a particular line of code, a yellow bar appears at the head of that line. After the file is saved, this line changes to green. Yellow code lines indicate that changes have been made but not yet saved to the file. Although you can't see the yellow bar next to lines 12 and 13 in the black-and-white screen shot shown in Figure 2-6, you may be able to see the shading difference at that point. The color difference indicates that these lines have recently been changed, as opposed to the green bar next to the rest of the lines of code.



Figure 2-6

Error notifications and assistance

In previous versions of Visual Studio, design-time error checking was a great feature of the IDE. As you typed your code, Visual Studio checked the code for errors. For instance, if you wrote an `If Then` statement (in Visual Basic) that didn't include an `End If` statement, the IDE would underline the `If Then` statement to remind you that the block of code was not complete. The line disappeared after you corrected the error. With Visual Studio 2005, if you make any design-time errors, a small square appears to the right of the underline (as shown under the `n` in `Then` in Figure 2-7).

```
Sub Button1_Click(ByVal sender
    Dim x As Integer = 3

    If x = 3 Then
End Sub
```

Figure 2-7

Hovering your cursor over the square causes an error sign to appear. Clicking the error sign opens up a dialog that gives you options for fixing the error. For example, if you are using an `If Then` statement without the closing `End If` statement in Visual Basic, clicking the error notification button provides you with a fix from the IDE, as shown in Figure 2-8.

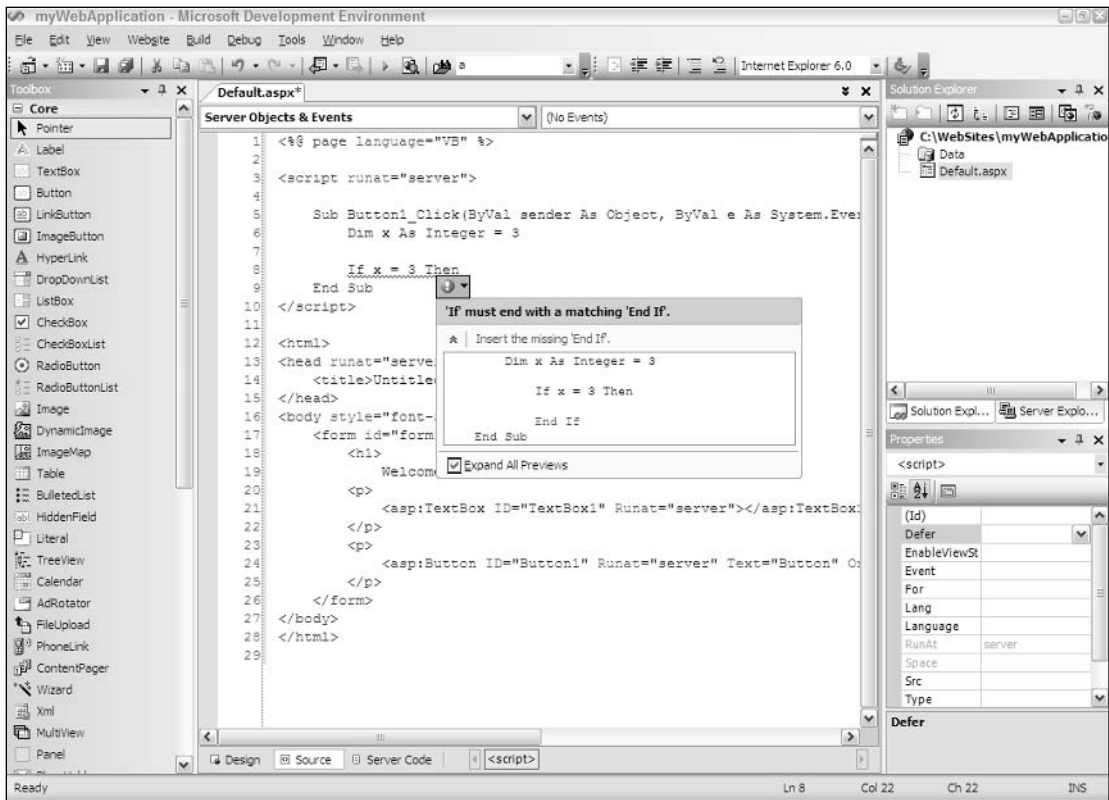


Figure 2-8

This pop-up dialog first states the issue. In this case, it says that any opening If statement must include a closing End If statement. Below this error notification is a link that enables you to apply the fix. Below the link is a code sample showing how the fix will affect your code.

Sometimes, more than one option exists for fixing a design-time error. For example, you might have the following code in your ASP.NET page:

```
Dim x As Integr
```

In this case, Integr is spelled incorrectly; the correct spelling, of course, is Integer. The IDE notifies you of this error and opens up the associated error dialog. You have three options for fixing the error (shown in Figure 2-9). To fix it, you simply scroll to the appropriate fix and click that link.

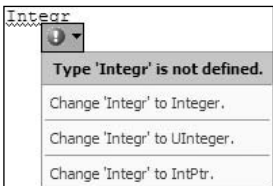


Figure 2-9

The Toolbox

One of the first changes you notice when you open this latest release of Visual Studio is a change in the Toolbox. The controls in the IDE are now presented in a hierarchical manner. This change was made because of the tremendous number of new controls in ASP.NET 2.0. The Toolbox is shown in Figure 2-10.



Figure 2-10

Because of the number of new controls (somewhere around 50), they have been organized into sections in the Toolbox. The new control sections include those shown in the following table.

Control Section	Controls the Section Contains
Core	This main control section contains the standard <code><asp: ></code> controls, such as TextBox, Button, and other core controls.
Data	Includes all the controls that deal with the retrieval and display of data that comes from a data store of some kind. Therefore, this section includes all the data source controls (SqlDataSource, AccessDataSource, and more), as well as the data display controls, such as GridView and DetailsView.
Personalization	Includes all the controls that deal with the new personalization features provided by ASP.NET 2.0, including all the WebPart controls such as WebPartManager and WebPartZone.
Security	Contains all controls that deal with adding user login and password capabilities to your ASP.NET applications, such as Login, LoginView, and LoginStatus.
Validation	Includes all the validation controls that have always been a part of ASP.NET, such as RequiredFieldValidator and RegularExpressionValidator.
Navigation	Includes controls that enable end users to work through a collection of ASP.NET pages, including SiteMapPath, Menu, and TreeView.
Crystal Reports	Includes all controls that enable users to work with Crystal Reports.
HTML	Includes the HTML server controls that have been a part of ASP.NET since the beginning. The names of these controls, however, have changed.
General	Contains only a pointer, although you are free to use this section for your own custom developed controls. (You can also create a completely new control section if you choose.)

One feature that has always been present in Visual Studio, but makes more sense now that so many new controls have been added, enables you to turn off the List View of the controls. Doing this causes the Toolbox to show the controls simply as icons (see Figure 2-11).

Right-click in the section of the Toolbox you want to change and deselect List View. This changes the view only for those controls in the section where you right-clicked. Each section in the Toolbox maintains its own settings.



Figure 2-11

The Solution Explorer

The Solution Explorer is still located where it was in previous versions of Visual Studio. The Solution Explorer, shown in Figure 2-12, provides you with an organized view of the projects in your application.



Figure 2-12

The toolbar at the top of the Solution Explorer still enables you to do many of the same tasks that you could perform in previous versions of Visual Studio, but this latest release of Visual Studio has some additional buttons on the toolbar. Figure 2-13 shows you the toolbar with a description of the items it contains.

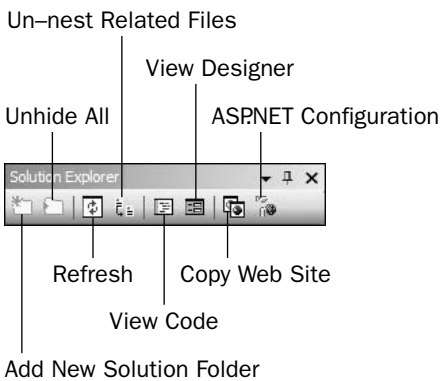


Figure 2-13

The Un-nest Related Files button is a new feature in the Solution Explorer that enables you to undo the nesting found in ASP.NET pages that are developed using code-behind files. By default when working with code-behind files, you click the plus sign next to the .aspx page to expose the code-behind file (.aspx.vb or .aspx.cs). Un-nesting these files puts them all on the same hierarchical level.

Another new button in the Solution Explorer is the Copy Web Site button. This opens up a new dialog in the Document Window that enables you to copy your application from one point to another. This dialog is shown in Figure 2-14.

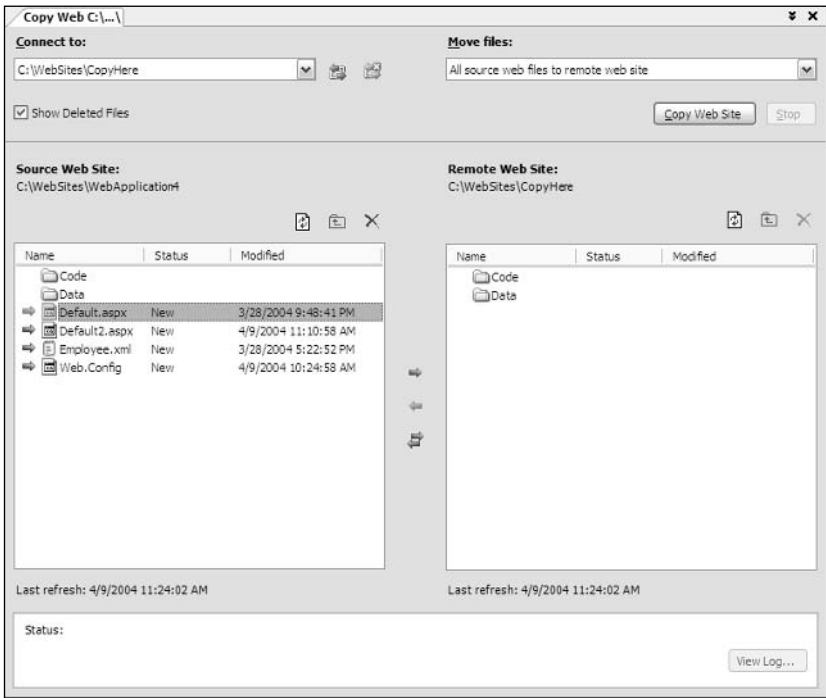


Figure 2-14

Using this dialog, you can copy your projects to a different place on the same server or to an entirely different server. You can now enjoy easy file movements and synchronization between two projects.

A final new button in the toolbar is the ASP.NET Configuration button that pulls up the ASP.NET configuration page for your selected application within the Document Window. This configuration system is discussed in detail in Chapter 14.

The Server Explorer

The Server Explorer is one of the more valuable windows within Visual Studio. This window can now be found on a separate tab next to the Solution Explorer. The Server Explorer (shown in Figure 2-15) enables you to perform a number of functions, such as working with database connectivity and performance monitoring and interacting with event logs.

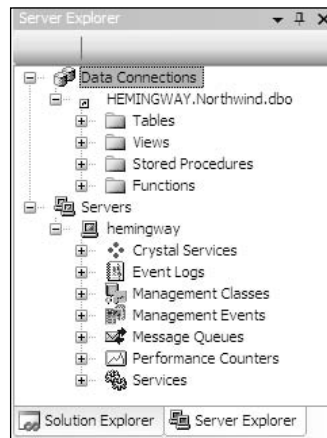


Figure 2-15

The Properties Window

The Properties Window is also relatively unchanged from the previous versions of Visual Studio. This window (shown in Figure 2-16) enables you to work with and control the properties of any item that is part of your application. After an item is selected, or if the cursor is focused on an item in the Code view of your ASP.NET page, the properties of that particular item are shown in the Properties Window.

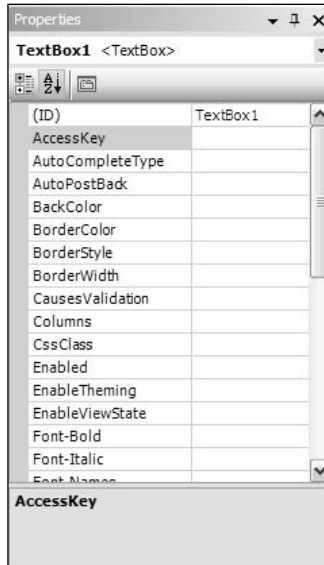


Figure 2-16

Lost Windows

You may not be able to find some familiar windows in the Visual Studio 2005 release. For instance, when you open one of your ASP.NET applications in Visual Studio 2005, you do not see the Class View and Dynamic Help windows. Although not apparent in the default view when the IDE first opens, these windows are still available for use with your applications.

You can find the Class View by choosing View ⇄ Other Windows ⇄ Class View from the Visual Studio menu. The Class View window opens directly next to the Server Explorer. You can move the window wherever you want within the IDE.

You can find the Dynamic Help window by choosing Help ⇄ Dynamic Help. Selecting this option opens the Dynamic Help window next to the Properties Window.

Other Common Visual Studio Activities

Visual Studio 2005 is so packed with functionality that it deserves a book of its own. This IDE is mammoth and enables you to do almost anything in the construction and management of your ASP.NET applications. This section takes a look at some of the common tasks that are done somewhat differently or in an altogether new manner in this latest release of Visual Studio.

Creating new projects

The process of creating new files and projects within Visual Studio 2005 is different than it was in Visual Studio 2002 or 2003. In this latest release of Visual Studio, the focus on project-based applications is gone. Now projects are created in a page-based manner. This means that when you create an ASP.NET application in Visual Studio, you don't find solution or project files. In fact, when you first create the application, the only items created for you by the IDE include the project folder and a single `.aspx` file. If you are creating an ASP.NET page using the code-behind model, you also have an `.aspx.vb` or `.aspx.cs` file.

One of the big changes you notice when opening the IDE is that no Start Page appears. You are presented with a blank IDE. You can create either a new single `.aspx` page or a Web site. To create a single page, simply go to the menu and choose File ⇨ New File. To work on a previous file, choose File ⇨ Open File. To create a new ASP.NET application, choose File ⇨ New Web Site. You can see the dialog of options in Figure 2-17.

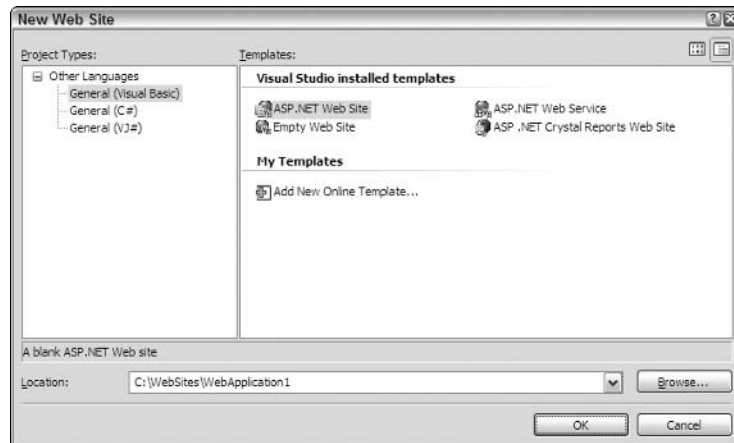


Figure 2-17

In most cases, you select the first option — ASP.NET Web Site. This creates a single folder for your application and a default `.aspx` page.

Making references to other objects

When you look at the Solution Explorer of your ASP.NET application, notice that the References and Web References folders are not present. How do you add these references to your file-based applications?

You can add them in a couple of ways, and both ways bring you to the same dialog within the IDE. The first way of adding references to your application is to highlight the solution in the Solution Explorer and then choose Web Site ⇨ References from the Visual Studio menu. The second option is to right-click the solution in the Solution Explorer and select Property Pages from the list of options. Both methods bring up the Property Pages dialog shown in Figure 2-18.

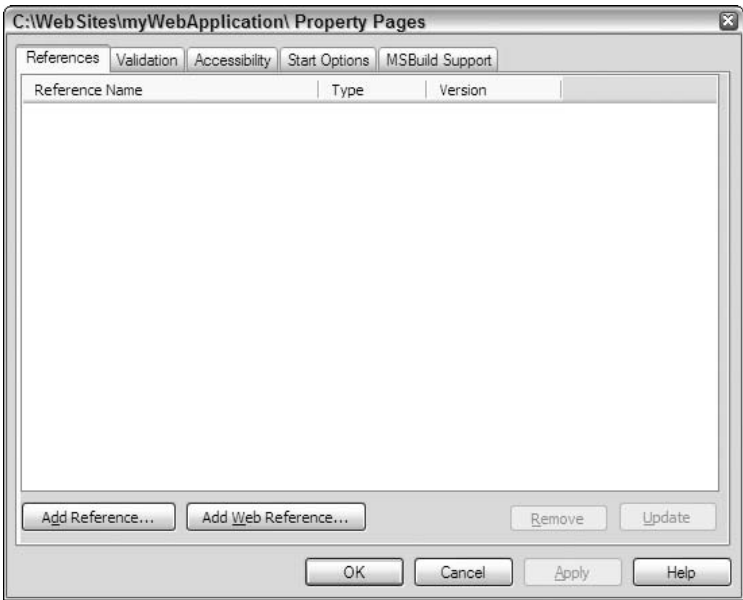


Figure 2-18

The Property Pages dialog allows you to make many modifications to your ASP.NET applications. For now, however, focus only on the first tab within the dialog — the References tab. When you have the References tab open, two enabled buttons appear at the bottom of the dialog — Add Reference and Add Web Reference.

The Add Reference button invokes the Add Reference dialog so that you can make a reference to a DLL to use in your project. Again in this version of Visual Studio, the objects are divided into categories such as .NET, COM, and others, as shown in Figure 2-19.

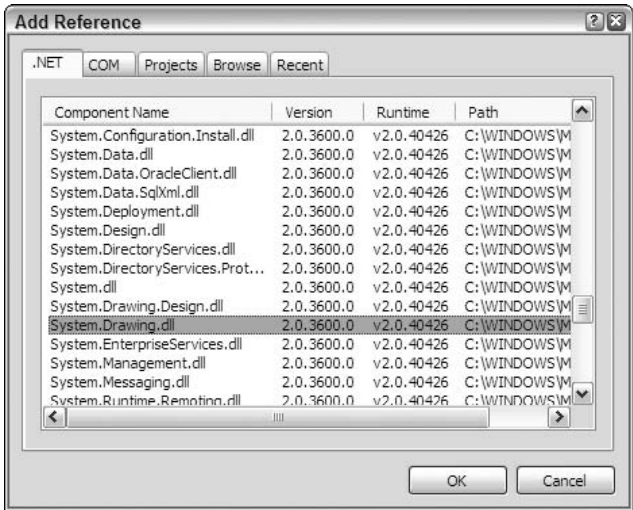


Figure 2-19

The Add Web References button invokes the Add Web Reference dialog (shown in Figure 2-20). Here you can make references to other Web services or .wsdl files found either in the same solution, on the same server, or on some remote server.

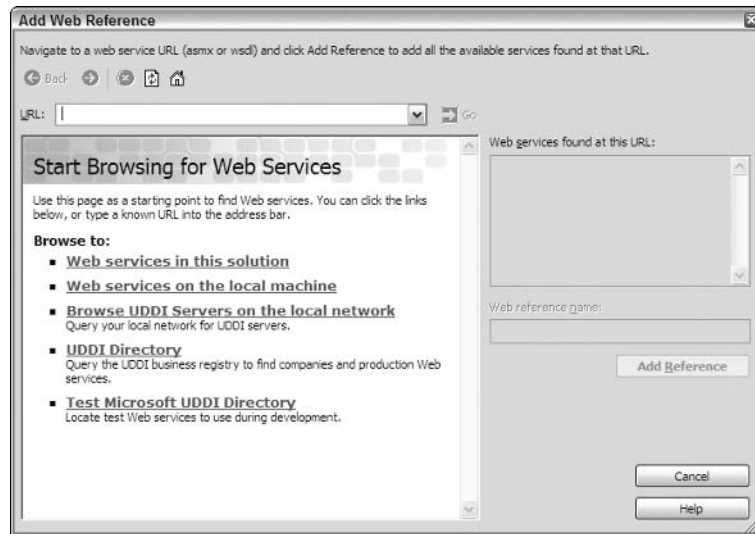


Figure 2-20

Be aware that these buttons have been added because no References or Web References folder appears in the Solution Explorer, which shows the referenced objects.

Using smart tags

The visual designer of Visual Studio now includes *smart tags*. Smart tags are a great enhancement to the development experience because they enable you to quickly program common tasks. Each smart tag is different and depends on the server control that it works with. For instance, the smart tag that appears for the GridView server control enables you to quickly apply paging and sorting of the data that the GridView displays. Other controls, however, may have different capabilities exposed through their respective smart tags.

Not every server control has a smart tag associated with it. If a server control does have this extra capability, you notice it after you drag and drop the control onto the design surface. After it is on the design surface, an arrow appears in the upper-right-hand corner of the control if a smart tag exists for that particular control. Clicking the arrow opens up the smart tag and all the options that the smart tag contains. This is illustrated in the GridView server control shown in Figure 2-21.

From the smart tag, you can select items either to add or alter by clicking one of the available links or by checking one of the available check boxes. When you have completed either of these actions, Visual Studio changes the code in the background — adding the capabilities that you want. You can also see the additions and modifications to the IDE if you change your view to the Code view of the page.

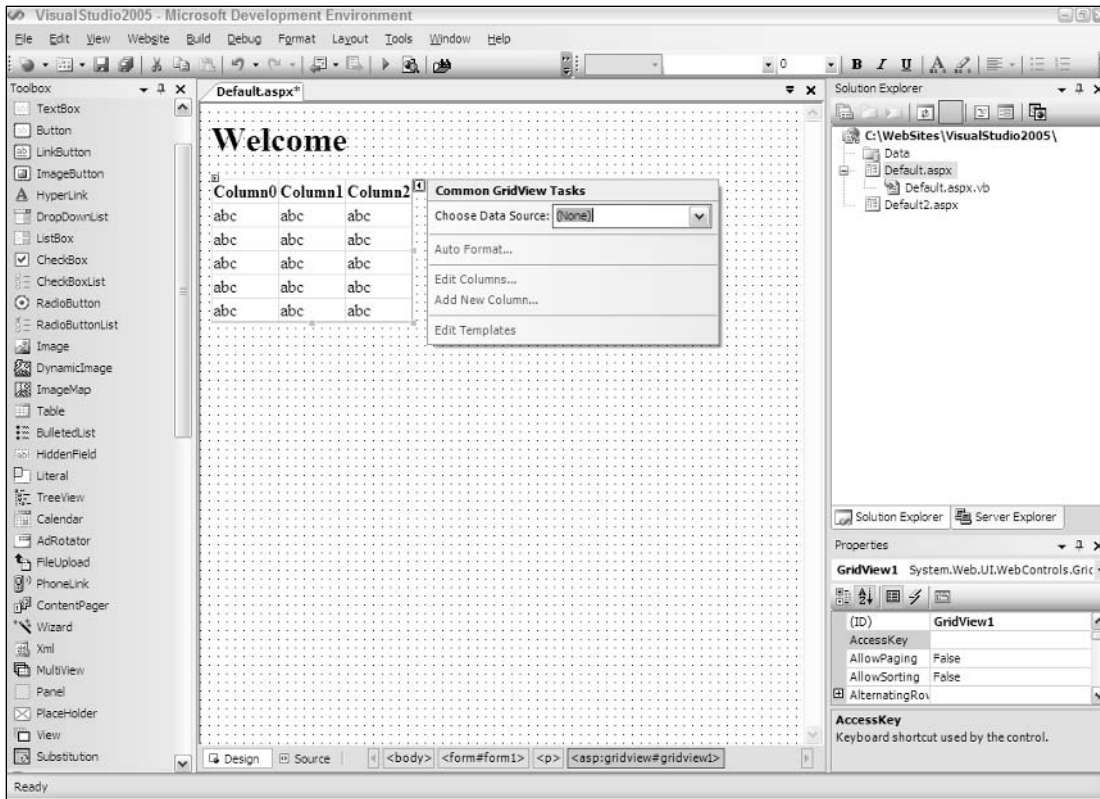


Figure 2-21

Saving and importing Visual Studio settings

Visual Studio 2005 allows for a tremendous number of customizations and modifications to the development environment and the development experience. You can do a lot to change Visual Studio either by dragging elements and components to new locations within the IDE, or by choosing Tools ⇨ Options in the Visual Studio menu bar to bring up the Options dialog shown in Figure 2-22.

The number of options that you can work with from this dialog are staggering and impossible to cover completely in this chapter. You have many of the same options that you worked with in the past, plus some new ones.

After you have Visual Studio set up as you want, you should save these settings so that they can be used again if you rebuild your computer, if you are working with a different instance of Visual Studio elsewhere, or if you want to share your settings with others. To save your settings, choose Tools ⇨ Import/Export Settings in the IDE. This pulls up the Import/Export Settings dialog shown in Figure 2-23.

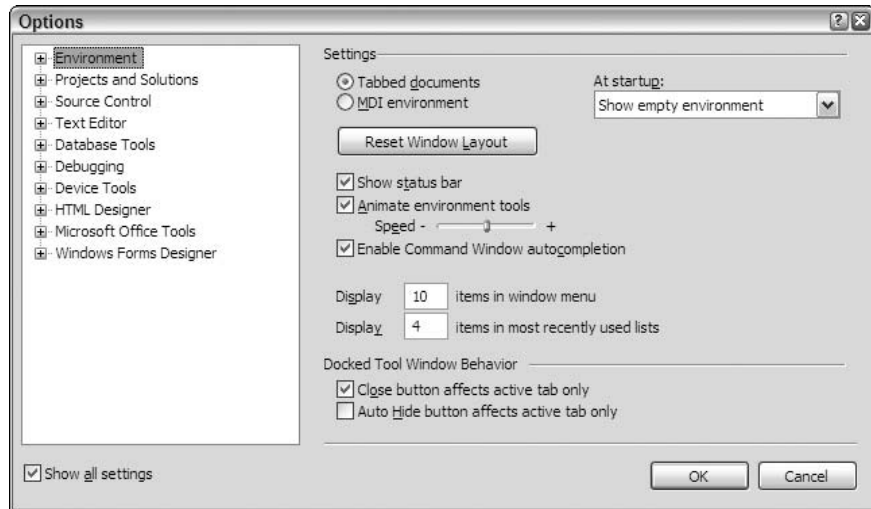


Figure 2-22

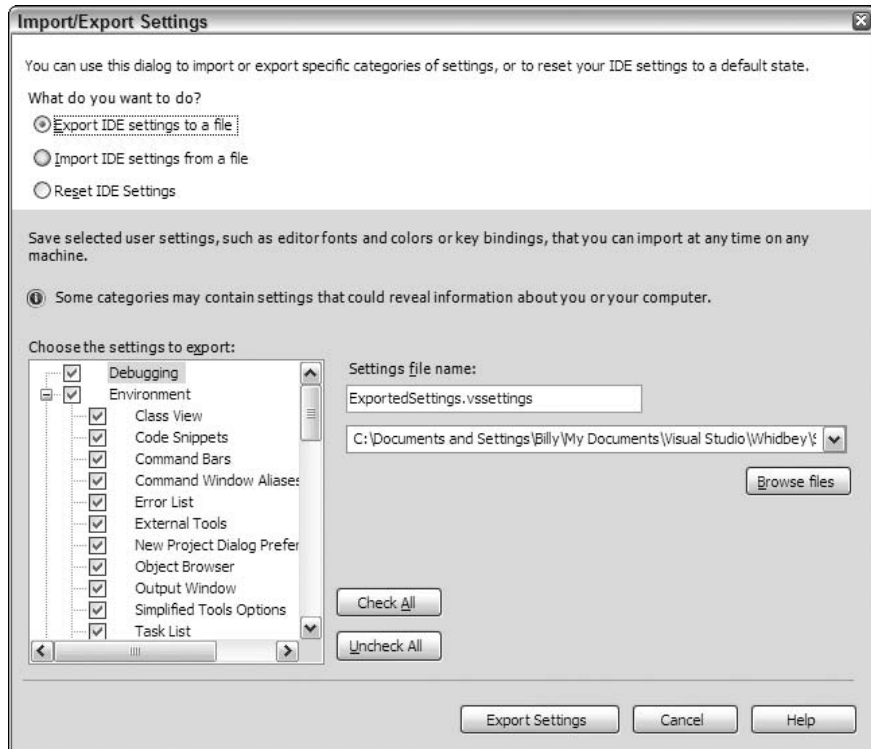


Figure 2-23

Chapter 2

From this dialog, you can either save your settings to a file that can be used elsewhere or you can import settings that are stored in the same type of file. You can also just reset Visual Studio to return the settings to the default that existed when Visual Studio was first installed and run.

If you are going to export your settings, select Export IDE Settings to a File. This shows a list of exportable settings in the left-hand pane of the dialog. By default, almost everything is selected. Feel free to uncheck the settings that you don't want to export. When this is set as you want, choose the name of the file and the location where you want to save the file. The file has a `.vssettings` extension. If you go back and look at the file, notice that Visual Studio saves the settings as an XML file.

Importing the settings is simply the process of making reference through the Import/Export Settings dialog to a file of the same type.

Summary

This chapter took a quick look at the best possible tool for creating ASP.NET 2.0 applications — Visual Studio 2005. This tool is unquestionably packed with functionality and makes you a more productive developer.

Included in this IDE are a number of wizards that make quick work of common programming tasks and allow you to concentrate on getting your applications live as soon as possible. Visual Studio 2005 expands on allowing developers to code to the database, to classes, and to the presentation layer — all from the same IDE.

This chapter was in no way meant to fully explain this IDE; my intention was to show you some of the newer features that you might utilize when building your applications. Delve more deeply into what is shown in the chapter, and you will find new features around every corner.

3

Application and Page Frameworks

When you first look at what is new in ASP.NET 2.0, you may be amazed by all the wonderful new server controls that it provides. You may marvel at how it enables you to work with data more effectively using the new data providers. You may be impressed at how easily you can build in security and personalization.

Its great capabilities don't end there, however. The application and ASP.NET pages as a whole also have some exciting new capabilities that you might overlook. This chapter takes a look at plenty of these new additions for working with ASP.NET pages and applications. One of the first steps you, the developer, should take when starting a project is to become familiar with the foundation you are building on and the options available for customizing this foundation.

Application Location Options

With ASP.NET 2.0, you now have the option — using Visual Studio 2005 — to create an application with a virtual directory mapped to IIS or a standalone application outside the confines of IIS. Whereas Visual Studio .NET forced developers to use IIS for all Web applications, Visual Studio 2005 includes a built-in Web server that you can use for development, much like you used the ASP.NET Web Matrix.

The following section shows you how to use this new built-in Web server that comes with ASP.NET 2.0.

Built-in Web server

By default, Visual Studio 2005 builds applications without the use of IIS. You can see this when you select New Web Site in the IDE. By default, the location provided for your application is in `C:\Websites\` (shown in Figure 3-1). It is not in `C:\inetpub\wwwroot\` as it would have been in

Chapter 3

Visual Studio .NET. Any site that you build and host inside C:\Websites\ (or any other folder you might create) uses the built-in Web server by default that is part of Visual Studio 2005. If you use the built-in Web server from Visual Studio 2005, you are not locked into the Websites folder; you can create any folder in your system that you want.

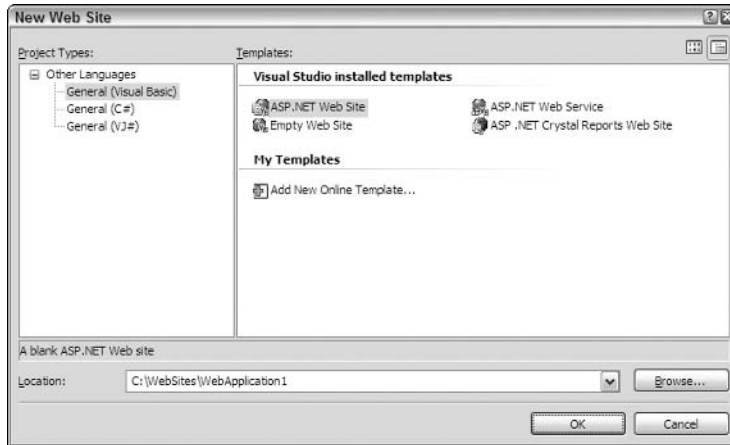


Figure 3-1

To change from this default, you have a handful of options. Click the Browse button in the New Web Site dialog. This brings up the Choose Location dialog, shown in Figure 3-2.

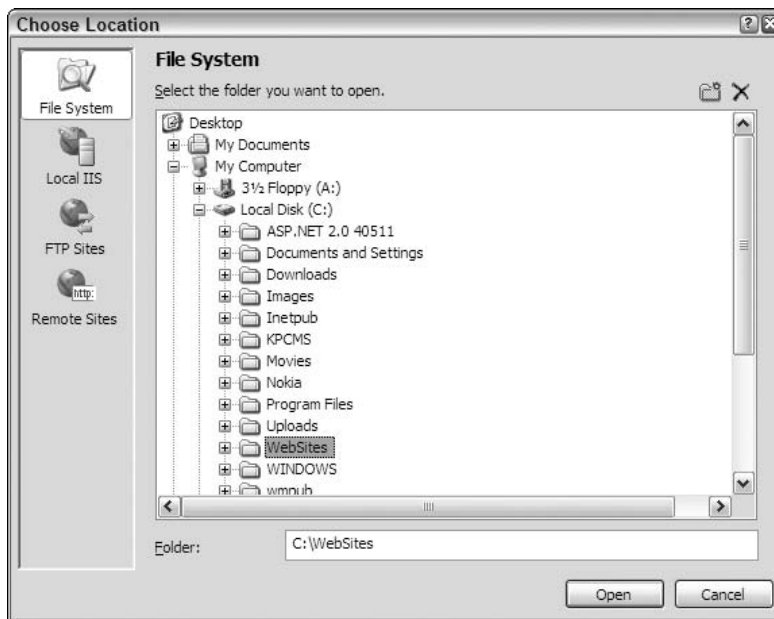


Figure 3-2

From this dialog, you can choose a new location for your Web application if you continue to use the built-in Web server that Visual Studio 2005 provides. To choose a new location, select a new folder and save your .aspx pages and any other associated files to this directory. When using Visual Studio 2005, you can run your application completely from this location. This new way of working with the ASP.NET pages you create is ideal for developers without access to a Web server, as it enables you to build applications that don't reside on a machine with IIS. This means that you can even develop ASP.NET applications on operating systems such as Windows XP Home Edition.

IIS

From the Choose Location dialog, you can also change where your application is saved and which type of Web server your application employs. To use IIS (as you probably did when you used Visual Studio .NET), select the Local IIS button in the dialog. This changes the results in the text area to show you a list of all the virtual application roots on your machine.

To create a new virtual root for your application, highlight Default Web Site. Two accessible buttons appear at the top of the dialog (see Figure 3-3). Looking from left to right, the first button in the upper-right corner of the dialog is for creating a new Web application — or a virtual root. This button is shown as a globe inside a box. The second button enables you to create virtual roots for any of the virtual directories you created. The third button is a Delete button, which allows you to delete any selected virtual directories or virtual roots on the server.

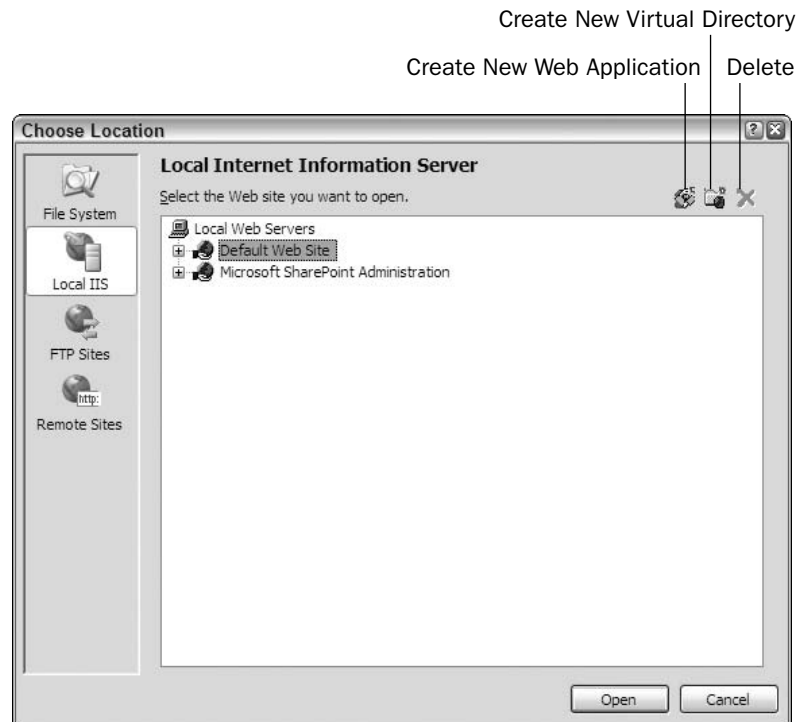


Figure 3-3

Chapter 3

After you have created the virtual directory you want, click the Open button. Visual Studio 2005 then goes through the standard process to create your application. Now, however, instead of depending on the built-in Web server from ASP.NET 2.0, your application uses IIS. When you invoke your application, the URL now contains something like `http://localhost/myweb/default.aspx` — meaning that it is using IIS.

FTP

Not only can you decide on the type of Web server for your Web application when you create it using the Choose Location dialog, but you can also decide where your application is going to be located. With the previous options, you built applications that resided on your local server. The FTP option enables you to actually store and even code your applications while they reside on a server somewhere else in your enterprise — or on the other side of the planet. You can also use the FTP capabilities to work on different locations within the same server. Using this new capability provides a wide range of possible options.

The built-in capability giving FTP access to your applications is a major enhancement to the IDE. Although formerly difficult to achieve, this is now quite simple, as you can see from Figure 3-4.

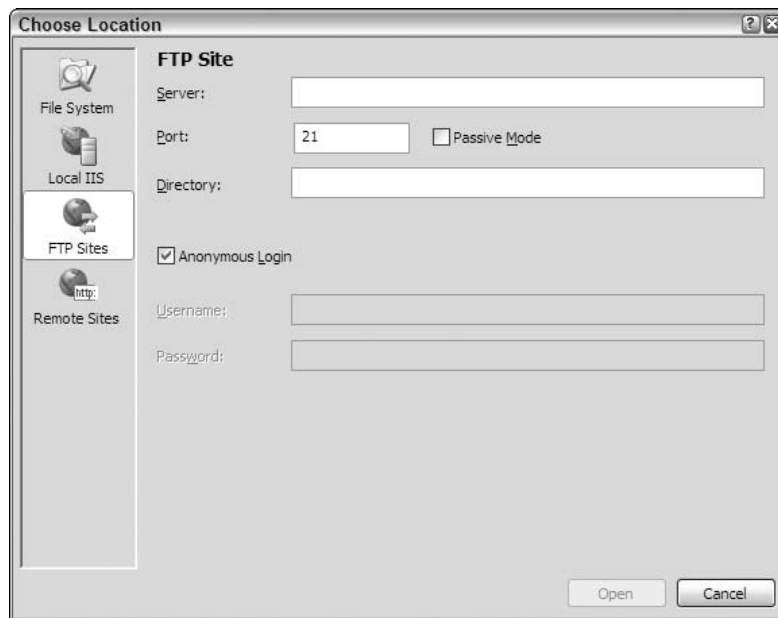


Figure 3-4

To create your application on a remote server using FTP, simply provide the server name, port to use, and the directory — as well as any required credentials. If the correct information is provided, Visual Studio 2005 reaches out to the remote server and creates the appropriate files for the start of your application, just as if it were doing the job locally. From this point on, you can open your project and connect to the remote server using FTP.

Web site requiring FrontPage Extensions

The last option in the Choose Location dialog is the Remote Sites option. Clicking this button provides a dialog that enables you to connect to a remote or local server that utilizes FrontPage Extensions. This option is displayed in Figure 3-5.

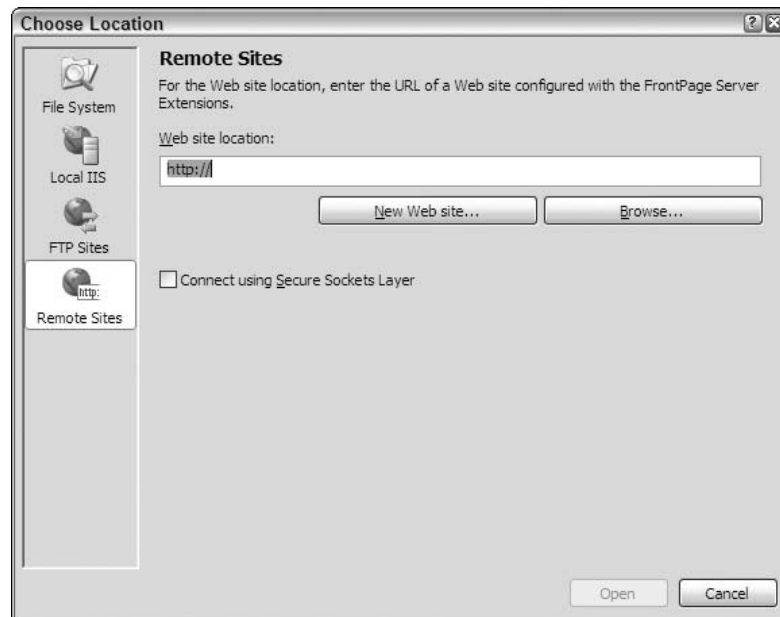


Figure 3-5

The ASP.NET Page Structure Options

One of the major complaints about Visual Studio .NET 2002 and 2003 is that it forced you to use the code-behind model when developing your ASP.NET pages. The code-behind model in ASP.NET was introduced as a new way to separate the presentation code and business logic. Listing 3-1 shows a typical .aspx page generated using Visual Studio .NET 2002 or 2003.

Listing 3-1: A typical .aspx page from ASP.NET 1.0/1.1

```
<%@ Page Language="vb" AutoEventWireup="false" Codebehind="WebForm1.aspx.vb"
    Inherits="WebApplication.WebForm1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
    <title>WebForm1</title>
    <meta name="GENERATOR" content="Microsoft Visual Studio .NET 7.1">
    <meta name="CODE_LANGUAGE" content="Visual Basic .NET 7.1">
    <meta name="vs_defaultClientScript" content="JavaScript">
    <meta name="vs_targetSchema" content="Microsoft.VisualStudio.Common.LanguageService.Schema">
</HEAD>
<BODY>
    <form id="Form1" runat="server">
    </form>
</BODY>
</HTML>
```

(continued)

Listing 3-1: *(continued)*

```
        content="http://schemas.microsoft.com/intellisense/ie5">
</HEAD>
<body>
    <form id="Form1" method="post" runat="server">
        <P>What is your name?<br>
        <asp:TextBox id="TextBox1" runat="server"></asp:TextBox><BR>
        <asp:Button id="Button1" runat="server" Text="Submit"></asp:Button></P>
        <P><asp:Label id="Label1" runat="server"></asp:Label></P>
    </form>
</body>
</HTML>
```

The code-behind file created within Visual Studio .NET 2002/2003 for the .aspx page is shown in Listing 3-2.

Listing 3-2: A typical .aspx.vb / .aspx.cs page from ASP.NET 1.0/1.1

```
Public Class WebForm1
    Inherits System.Web.UI.Page

    #Region " Web Form Designer Generated Code "

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

    End Sub
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label

    'NOTE: The following placeholder declaration is required by the Web Form
    Designer.
    'Do not delete or move it.
    Private designerPlaceholderDeclaration As System.Object

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

    #End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles Button1.Click
        Label1.Text = "Hello " & TextBox1.Text
    End Sub
End Class
```

In this code-behind page from ASP.NET 1.0/1.1, you can see that a lot of code developers never have to deal with is hidden in the `Region` section of the page. Because ASP.NET 2.0 is built on top of .NET 2.0, it can now take advantage of the new .NET Framework capability of partial classes. Partial classes enable you to separate your classes into multiple class files, which are then combined into a single class when the application is compiled. Because ASP.NET 2.0 combines all this page code for you behind the scenes when the application is compiled, the code-behind files you work with in ASP.NET 2.0 are simpler in appearance and the model is easier to use. You are presented with only the pieces of the class that you need. Now take a look at both the inline and code-behind models from ASP.NET 2.0.

Inline coding

In the past, many developers chose to develop against Visual Studio .NET and built their ASP.NET pages inline. Now Visual Studio 2005 allows you to build using this coding style. To build an ASP.NET page inline instead of using the code-behind model, you simply select the page type from the Add New Item dialog (see Figure 3-6) and uncheck the Place Code in Separate File check box. You can get at this dialog by right-clicking on the project or the solution in the Solution Explorer and selecting Add New Item.

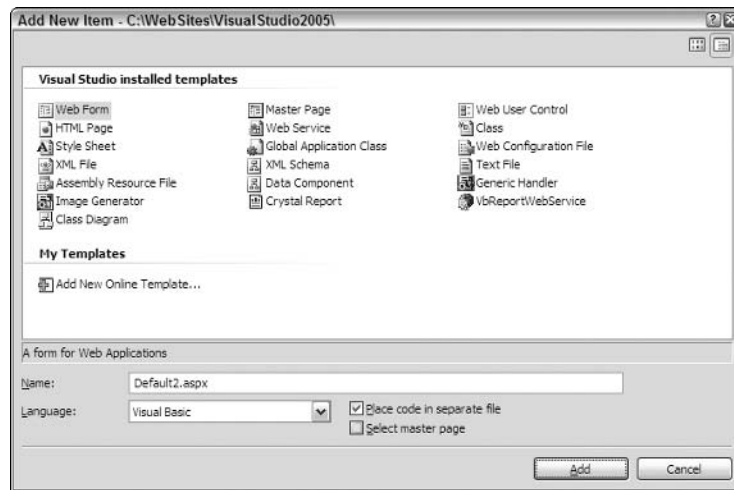


Figure 3-6

From here, you can see the check box you need to unselect if you want to build your ASP.NET pages inline. In fact, many page types have options for both inline and code-behind styles. The following table shows your inline options when selecting files from this dialog.

File Options Using Inline Coding	Option Creates
Web Form	.aspx file
Master Page	.master file
Web User Control	.ascx file
Web Service	.asmx file

Chapter 3

By using the Web Form option with a few controls, you get a page that encapsulates not only the presentation logic, but the business logic as well. This is illustrated in Listing 3-3.

Listing 3-3: A simple page that uses the inline coding model

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & Textbox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Simple Page</title>
</head>
<body>
    <form runat="server">
        What is your name?<br />
        <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
        <asp:Button ID="Button1" Runat="server" Text="Submit"
            OnClick="Button1_Click" />
        <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + Textbox1.Text;
    }
</script>
```

From this example, you can see that all the business logic is encapsulated in between `<script>` tags. The nice feature of the inline model is that the business logic and the presentation logic are contained within the same file. Some developers find that having everything in a single viewable instance makes working with the ASP.NET page easier. The other great thing about the inline coding model and ASP.NET 2.0 is that Visual Studio 2005 now provides IntelliSense when working with these types of files. In the past, this capability didn't exist. Visual Studio forced you to use the code-behind model and, even if you rigged it so your pages were using the inline model, you lost all IntelliSense capabilities.

New code-behind model

The other option for constructing your ASP.NET 2.0 pages is to build your files using the new code-behind model. I say *new* because, even though the idea of the code-behind model is the same as it was in previous versions of ASP.NET, the way in which the code-behind model is used in ASP.NET 2.0 is quite a bit different.

To create a new page in your ASP.NET solution that uses the code-behind model, select the page type you want from the Add New Item dialog. Just as many of the pages options have inline options, there also are code-behind file options in this dialog. To build a page using the code-behind model, you have to select the page in the Add New Item dialog and check the Place Code in Separate File check box. The following table shows you the options for pages that use the code-behind model.

File Options Using Code-Behind	Option Creates
Web Form	.aspx file .aspx.vb or .aspx.cs file
Master Page	.master file .master.vb or .master.cs file
Web User Control	.ascx file .ascx.vb or .ascx.cs file
Web Service	.asmx file .asmx.vb or .asmx.cs file

The idea of using the code-behind model is to separate the business logic and presentation logic into separate files. Doing this makes it easier to work with your pages, especially if you are working in a team environment where visual designers work on the UI of the page and coders work on the business logic that sits behind the presentation pieces. In the earlier Listings 3-1 and 3-2, you saw how pages using the code-behind model in ASP.NET 1.0/1.1 were constructed. To see the difference in ASP.NET 2.0, take a look at how its code-behind pages are constructed. This is illustrated in Listing 3-4 for the presentation piece and Listing 3-5 for the code-behind piece.

Listing 3-4: An .aspx page that uses the ASP.NET 2.0 code-behind model

VB

```
<%@ Page Language="VB" AutoEventWireup="false" CompileWith="Default.aspx.vb"
    ClassName="Default_aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Simple Page</title>
</head>
<body>
```

(continued)

Listing 3-4: *(continued)*

```
<form runat="server">
    What is your name?<br />
    <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox><br />
    <asp:Button ID="Button1" Runat="server" Text="Submit"
        OnClick="Button1_Click" />
    <p><asp:Label ID="Label1" Runat="server"></asp:Label></p>
</form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" CompileWith="Default.aspx.cs" ClassName="Default_aspx" %>
```

Listing 3-5: A code-behind page**VB**

```
Imports Microsoft.VisualBasic

Partial Class Default_aspx

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text
    End Sub

End Class
```

C#

```
using System;
using System.Configuration;
using System.Web;
using System.Web.Caching;
using System.Web.SessionState;
using System.Web.Security;
using System.Web.Profile;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class Default_aspx
{
    void Button1_Click(object sender, EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text;
    }
}
```

The .aspx page using this new ASP.NET 2.0 code-behind model has some attributes in the Page directive different from those you are used to. The first is the `CompileWith` attribute. This is a new attribute in the Page directive and is meant to point to the code-behind page that is used with this presentation

page. In this case, the value assigned is `Default.aspx.vb` or `Default.aspx.cs`. The second attribute needed is the `ClassName` attribute. This attribute was available in previous versions of ASP.NET, but was little used. This attribute specifies the name of the class that is bound to the page when the page is compiled. The directives are simple enough in ASP.NET 2.0. Take a look at the code-behind page from Listing 3-5.

The code-behind page is rather simple in appearance now using the partial class capabilities that .NET 2.0 provides. You can see that the class created in the code-behind file uses partial classes, employing the new `Partial` keyword in Visual Basic 8.0 and the `partial` keyword from C#. This enables you to simply place the methods that you need in your page class. In this case, you have a button-click event and nothing else.

New Page Directives

ASP.NET directives are something that is a part of every ASP.NET page. You can control the behavior of your ASP.NET pages by using these directives. Here's an example of the `Page` directive:

```
<%@ Page Language="VB" AutoEventWireup="false" CompileWith="Default.aspx.vb"
    ClassName="Default_aspx" %>
```

New attributes

These page directives are commands for the compiler to use as the page is compiled. A large number of attributes have always been available to the `Page` directive itself, but with the introduction of ASP.NET 2.0, some additional attributes are available. These six important new attributes are explained in the following table.

New Attribute	Description
<code>CompileWith</code>	Takes a <code>String</code> value which points to the code-behind file used.
<code>EnablePersonalization</code>	<code>Boolean</code> value that specifies whether the new ASP.NET 2.0 personalization features are used with the page.
<code>LinePragmas</code>	<code>Boolean</code> value that specifies whether line pragmas are used with the resulting assembly.
<code>Master</code>	Takes a <code>String</code> value that points to the location of the master page used with the page. This attribute is used with content pages.
<code>PersonalizationProvider</code>	Takes a <code>String</code> value that specifies the name of the personalization provider used in applying personalization to the page.
<code>Theme</code>	<code>String</code> value that specifies the theme used with the page.

New directives

In addition to the new attributes used with the Page directive, two new directives can be also be used for the pages you create. These new directives include the following:

- ❑ Master: This directive is used in the creation of master pages — the main templates used by any subpages within your applications. Master pages are explained in Chapter 6.
- ❑ PreviousPage: This directive is used to specify the page from which any cross-postings originate. Cross-posting ASP.NET pages are explained later in this chapter.

The Master page directive is similar to the Page directive, but you specify properties of the templated page that you will be using in conjunction with any number of content pages in your site. Any content pages (built using the Page directive) you might have can then inherit from the master page all the master content (defined in the master page using the Master directive). Although they are similar, the Master directive has fewer attributes available to it than the Page directive. The available attributes for the Master directive include:

- ❑ AutoEventWireUp
 - ❑ ClassName
 - ❑ CodeBehind
 - ❑ CompilerOptions
 - ❑ CompileWith
 - ❑ Debug
 - ❑ Description
 - ❑ EnablePersonalization
 - ❑ EnableViewState
- ❑ Explicit
 - ❑ Inherits
 - ❑ Language
 - ❑ LinePragmas
 - ❑ Master
 - ❑ Src
 - ❑ Strict
 - ❑ Theme

These attributes need not be defined because they are the same as those for the Page directive, except that they apply to the master page that is used in templating your pages.

The PreviousPage directive is a new directive that works with the new cross-page posting capability that ASP.NET 2.0 provides. This simple directive contains only two possible attributes: TypeName and VirtualPath. The following table describes these two new attributes.

New Attribute	Description
TypeName	Specifies the strong type used in the previous page.
VirtualPath	String value specifying the relative path of the page that is cross-posting to the working page.

New Page Events

ASP.NET developers consistently work with various events in their server-side code. Many of the events that they work with pertain to specific server controls. For instance, if you want to initiate some action when the end-user clicks a button on your Web page, you create a button-click event in your server-side code, as shown in Listing 3-6.

Listing 3-6: A sample button-click event shown in VB

```
Sub Button1_Click(sender As Object, e As EventArgs)
    Label1.Text = TextBox1.Text
End Sub
```

In addition to the server controls, developers also want to initiate actions at specific moments when the ASP.NET page is being either created or destroyed. The ASP.NET page itself has always had a number of events that you could work with for these instances. The following list shows you all the page events you could have used in ASP.NET 1.0/1.1:

- | | |
|--|------------------------------------|
| <input type="checkbox"/> AbortTransaction | <input type="checkbox"/> Init |
| <input type="checkbox"/> CommitTransaction | <input type="checkbox"/> Load |
| <input type="checkbox"/> DataBinding | <input type="checkbox"/> PreRender |
| <input type="checkbox"/> Disposed | <input type="checkbox"/> Unload |
| <input type="checkbox"/> Error | |

One of the more popular page events from this list is the `Load` event, which is used in VB as shown in Listing 3-7.

Listing 3-7: Using the Page_Load event

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

    Response.Write("This is the Page_Load event")
End Sub
```

Besides the page events just shown, ASP.NET 2.0 adds the following new events:

- ☐ `InitComplete`: For when the initialization of the page is completed.
- ☐ `LoadComplete`: For when the page has been completely loaded into memory.
- ☐ `PreInit`: For the moment directly before a page has been initialized.
- ☐ `PreLoad`: For the moment before a page has been loaded into memory.
- ☐ `PreRenderComplete`: For the moment directly before a page has been rendered in the browser.

You construct these new page events just as you did the previously shown page events. For example, you use the `PreInit` event as shown in Listing 3-8.

Listing 3-8: Using the new page events

VB

```
<script runat="server" language="vb">
    Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
        Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    void Page_PreInit(object sender, System.EventArgs e)
    {
        Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

If you create an ASP.NET 2.0 page and turn on tracing, you can see the order in which the main page events are initiated. They are fired in the following order:

- ☐ PreInit
- ☐ Init
- ☐ InitComplete
- ☐ PreLoad
- ☐ Load
- ☐ LoadComplete
- ☐ PreRender
- ☐ PreRenderComplete

With the addition of these new choices, you can now work with the page and the controls on the page at many different points in the page-compilation process. You see these useful new page events in code examples throughout the book.

Cross-Page Posting

One common feature in ASP 3.0 that is difficult to achieve in ASP.NET 1.0/1.1 is the capability to do cross-page posting. Cross-page posting enables you to submit a form (say, `Page1.aspx`) and have this form and all the control values post themselves to another page (`Page2.aspx`).

Traditionally, any page created in ASP.NET 1.0/1.1 simply posted to itself, and you handled the control values within this page instance. You could differentiate between the page's first request and any post-backs by using the `Page.IsPostBack` property, as shown here:

```
If Page.IsPostBack Then
    ' deal with control values
End If
```

Even with this capability, many developers still wanted to be able to post to another page and deal with the first page's control values on that page. This is now possible in ASP.NET 2.0, and it is quite simple to achieve as well.

For an example, create a page called `Page1.aspx` that contains a simple form. This page is shown in Listing 3-9.

Listing 3-9: Page1.aspx

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text & "<br />" & _
            "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>First Page</title>
</head>
<body>
    <form id="form1" runat="server">
        Enter your name:<br />
        <asp:Textbox ID="TextBox1" Runat="server">
        </asp:Textbox>
        <p>
        When do you want to fly?<br />
        <asp:Calendar ID="Calendar1" Runat="server"></asp:Calendar></p>
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit page to itself"
            OnClick="Button1_Click" />
        <asp:Button ID="Button2" Runat="server" Text="Submit page to Page2.aspx"
           PostBackUrl="Page2.aspx" />
        <p>
        <asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>
```

C#

```
<%@ page language="C#" %>

<script runat="server">
    void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```


Chapter 3

The code from `Page1.aspx`, as shown in Listing 3-9, is quite interesting. First, two buttons are shown on the page. Both buttons submit the form, but each submits the form to a different location. The first button submits the form to itself. This is the behavior that has been the default for ASP.NET 1.0/1.1. In fact, there is nothing different about `Button1`. It submits to `Page1.aspx` as a postback because of the use of the `OnClick` property in the button control. A `Button1_Click` event on `Page1.aspx` handles the values that are contained within the server controls on the page.

The second button, `Button2`, works quite differently. This button does not contain an `OnClick` event as the first button did. Instead, it uses the `PostBackUrl` property. This property takes a string value that points to the location of the file that this page should post to. In this case, it is `Page2.aspx`. This means that `Page2.aspx` now receives the postback and all the values contained in the `Page1.aspx` controls. Look at the code for `Page2.aspx`, shown in Listing 3-10.

Listing 3-10: Page2.aspx

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim pp_Textbox1 As TextBox
        Dim pp_Calendar1 As Calendar

        pp_Textbox1 = CType(PreviousPage.FindControl("Textbox1"), TextBox)
        pp_Calendar1 = CType(PreviousPage.FindControl("Calendar1"), Calendar)

        Label1.Text = "Hello " & pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Second Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:label id="Label1" runat="server"></asp:label>
    </form>
</body>
</html>
```

C#

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
```

```

        TextBox pp_Textbox1;
        Calendar pp_Calendar1;

        pp_Textbox1 = (TextBox)PreviousPage.FindControl("Textbox1");
        pp_Calendar1 = (Calendar)PreviousPage.FindControl("Calendar1");

        Label1.Text = "Hello " + pp_Textbox1.Text + "<br />" + "Date Selected: " +
            pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>

```

You have a couple of ways of getting at the values of the controls that are exposed from `Page1.aspx` from the second page. The first option is displayed in Listing 3-10. To get at a particular control's value that is carried over from the previous page, you simply create an instance of that control type and populate this instance using the `FindControl` method from the `PreviousPage` property. The `String` value assigned to the `FindControl` method is the `Id` value, which is used for the server control from the previous page. After this is assigned, you can work with the server control and its carried-over values as if it resided on the current page to begin with. You can see from the example that you can extract the `Text` and `SelectedDate` properties from the controls without any problems.

Another way of exposing the control values from the first page (`Page1.aspx`) is to create a `Property` for the control. This is shown in Listing 3-11.

Listing 3-11: Exposing the values of the control from a Property

VB

```

<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Public ReadOnly Property pp_TextBox1() As TextBox
        Get
            Return TextBox1
        End Get
    End Property

    Public ReadOnly Property pp_Calendar1() As Calendar
        Get
            Return Calendar1
        End Get
    End Property

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text & "<br />" & _
            "Date Selected: " & Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>

```

(continued)

Listing 3-11: *(continued)*

C#

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    public TextBox pp_TextBox1
    {
        get
        {
            return TextBox1;
        }
    }

    public Calendar pp_Calendar1
    {
        get
        {
            return Calendar1;
        }
    }

    void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text + "<br />" +
            "Date Selected: " + Calendar1.SelectedDate.ToShortDateString();
    }
}
</script>
```

Now that these properties are exposed on the posting page, the second page (`Page2.aspx`) can work with the server control properties that are exposed from the first page in an easier fashion. Listing 3-12 shows you how `Page2.aspx` works with these exposed properties.

Listing 3-12: Consuming the exposed properties from the first page

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" & _
            "Date Selected: " & _
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + PreviousPage.pp_TextBox1.Text + "<br />" +
            "Date Selected: " +
            PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
    }
</script>
```

In order to be able to work with the properties that `Page1.aspx` exposes, you have to strongly type the `PreviousPage` property to `Page1.aspx`. To do this, you use the `PreviousPageType` directive. This new directive allows you to specifically point to `Page1.aspx` with the use of the `VirtualPath` attribute. When that is in place, notice that you can see the properties that `Page1.aspx` exposes through IntelliSense from the `PreviousPage` property. This is illustrated in Figure 3-7.

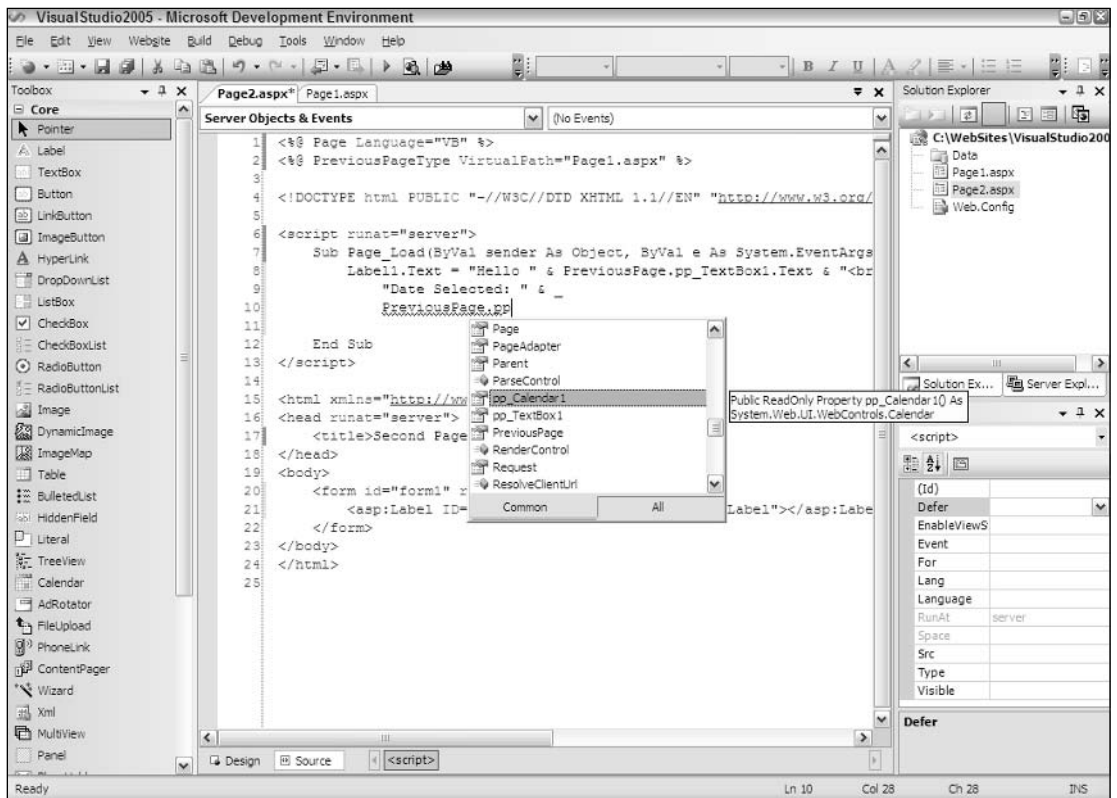


Figure 3-7

As you can see, working with cross-page posting is pretty straightforward. One last point is that when you are cross-posting from one page to another, you don't need to work with the postback on the second page only. In fact, you can still create methods on `Page1.aspx` that work with postback before moving onto `Page2.aspx`. To do this, you simply add an `OnClick` event for the button in `Page1.aspx` and a method along with assigning a value for the `PostBackUrl` property. You can then work with the postback on `Page1.aspx` and then again on `Page2.aspx`.

What happens if someone requests `Page2.aspx` before working through `Page1.aspx`? It is actually quite easy to work with the request to see if it is coming from `Page1.aspx` or if someone just hit `Page2.aspx` directly. You can work with the request through the use of the `IsCrossPagePostBack` property. Quite similar to the `IsPostBack` property that you are used to from ASP.NET 1.0/1.1, the `IsCrossPagePostBack` property enables you to check whether the request is from `Page1.aspx`. Listing 3-13 shows an example of this.

Listing 3-13: Using the `IsCrossPagePostBack` property

VB

```
<%@ Page Language="VB" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Page.IsCrossPagePostBack Then
            Label1.Text = "Hello " & PreviousPage.pp_Textbox1.Text & "<br />" & _
                "Date Selected: " & _
                PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString()
        Else
            Response.Redirect("Page1.aspx")
        End If
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" %>
<%@ PreviousPageType VirtualPath="Page1.aspx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        if (Page.IsCrossPagePostBack) {
            Label1.Text = "Hello " + PreviousPage.pp_Textbox1.Text + "<br />" +
                "Date Selected: " +
                PreviousPage.pp_Calendar1.SelectedDate.ToShortDateString();
        }
        else
        {
            Response.Redirect("Page1.aspx");
        }
    }
</script>
```

New Application Folders

When you create ASP.NET applications, notice that ASP.NET 2.0 now uses a file-based approach. When working with ASP.NET 2.0, you can have as many files and folders within your application as you can build without recompiling the application each and every time a new file is added to the overall solution. ASP.NET 2.0 now includes the capability to automatically precompile your ASP.NET applications dynamically.

ASP.NET 1.0/1.1 compiled everything in your solution into a DLL. This is no longer necessary because ASP.NET 2.0 applications have a defined folder structure. By using the ASP.NET 2.0 defined folders, you can have your code automatically compiled for you, your application themes accessible throughout your application, and your globalization resources available whenever you need them. Take a look at each of these defined folders to see how they work. The first is the `\Code` folder.

`\Code` folder

The `\Code` folder is meant to store your classes, `.wsdl` files, and typed datasets. Any of these items stored in this folder are then automatically available to all the pages within your solution. The nice thing about the `\Code` folder is that when you place something inside this folder, Visual Studio 2005 automatically detects this and compiles it if it is a class (`.vb` or `.cs`), automatically creates your XML Web service proxy class (from the `.wsdl` file), or automatically creates a typed dataset for you from your `.xsd` files. After the files are automatically compiled, these items are then instantaneously available to use in any of your ASP.NET pages that are in the same solution. Look at how to employ a simple class in your solution using the `\Code` folder.

The first step is to create a `\Code` folder. To do this, simply right-click the solution and select **New Folder**. Name the folder `Code`. Right away you notice that Visual Studio 2005 treats this folder differently from the other folders in your solution. The `Code` folder is shown in a different color (gray) with a document pictured next to the folder icon. See Figure 3-8.

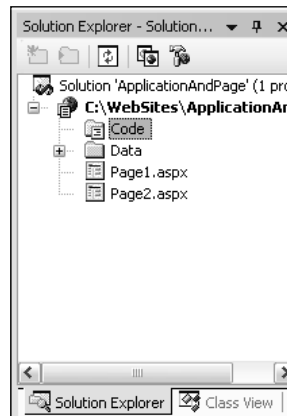


Figure 3-8

After the `\Code` folder is in place, right-click the folder and select **Add New Item**. The **Add New Item** dialog that appears doesn't give you many options for the types of files that you can place within this folder. The available options include a **Class** file, an **XML Schema**, and an **Assembly Resource File**. For this first example, select **Class** and name the class `Calculator.vb` or `Calculator.cs`. Listing 3-13 shows how the `Calculator` class should appear.

Listing 3-13: The Calculator class

VB

```
Imports Microsoft.VisualBasic

Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function
End Class
```

C#

```
using System;

public class Calculator
{
    public int Add(int a, int b)
    {
        return (a + b);
    }
}
```

What's next? Just save this file, and it is now available to use in any pages that are in your solution. To see this in action, create a simple .aspx page that has just a single Label server control. Listing 3-14 shows you the code to place within the Page_Load event to use this new class available to the page.

Listing 3-14: An .aspx page that uses the Calculator class

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myCalc As New Calculator
        Label1.Text = myCalc.Add(12, 12)
    End Sub
</script>
```

C#

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        Calculator myCalc = new Calculator();
        Label1.Text = myCalc.Add(12, 12).ToString();
    }
</script>
```

When you run this .aspx page, notice that it utilizes the Calculator class without any problem, with no need to compile the class before use. In fact, right after saving the Calculator class in your solution or moving the class to the \Code folder, you also instantaneously receive IntelliSense capability on the methods that the class exposes (as illustrated in Figure 3-9).

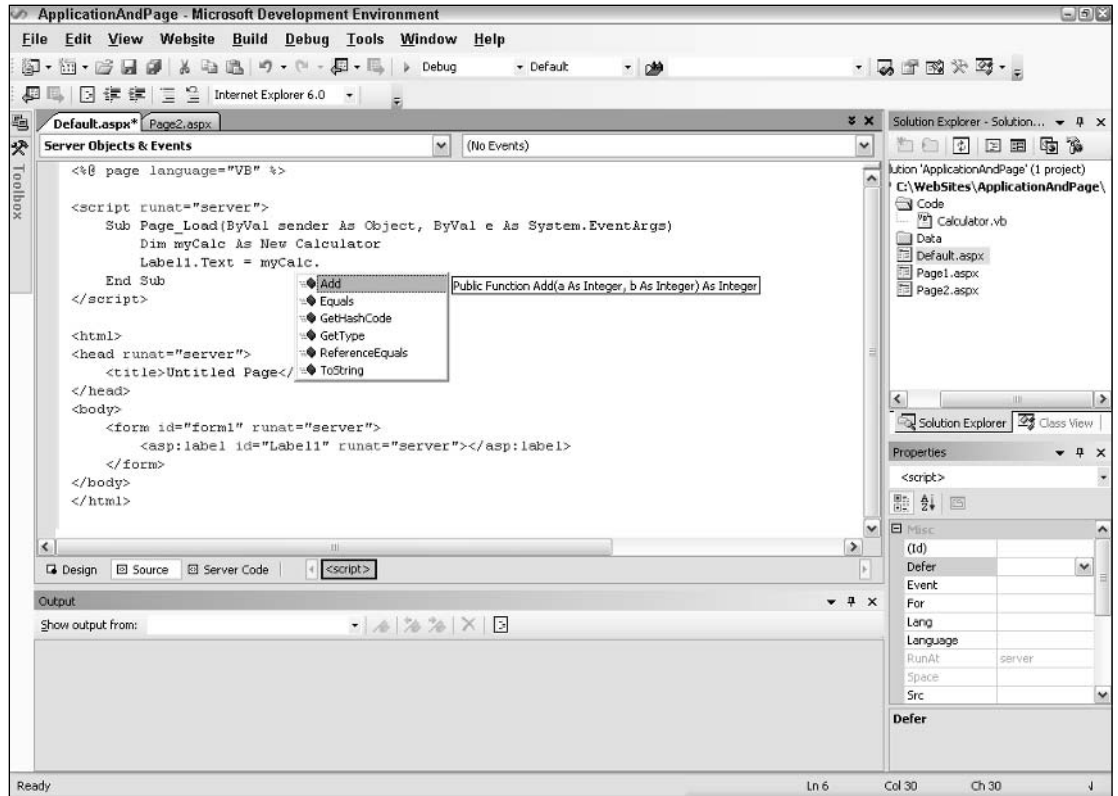


Figure 3-9

To see Visual Studio 2005 works with the \Code folder, open the Calculator class again in the IDE and add a Subtract method. Your class should now appear as shown in Listing 3-15.

Listing 3-15: Adding a Subtract method to the Calculator class

```
VB
Imports Microsoft.VisualBasic

Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return (a + b)
    End Function
```

(continued)

Listing 3-15: *(continued)*

```
Public Function Subtract(ByVal a As Integer, ByVal b As Integer) As Integer
    Return (a - b)
End Function
End Class
```

C#

```
using System;
```

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return (a + b);
    }
}
```

```
    public int Subtract(int a, int b)
    {
        return (a - b);
    }
}
```

After you have added the `Subtract` method to the `Calculator` class, save the file and go back to your `.aspx` page. Notice that the class has been recompiled by the IDE and the new method is now available to your page. You see this directly in IntelliSense. Figure 3-10 shows this in action.

Everything placed in the `\Code` folder is compiled into a single assembly. The class files placed within the `\Code` folder are also not required to use a specific language. This means that even if all the pages of the solution are in Visual Basic 8.0, the `Calculator` class in the `\Code` folder of the solution could be built in C# (`Calculator.cs`).

Because all the classes contained in this folder are built into a single assembly, you cannot have classes of different languages sitting in the root `\Code` folder, as in the following:

```
\Code
    Calculator.cs
    AdvancedMath.vb
```

Having two classes made up of different languages in the `\Code` folder (as shown here) causes an error to be thrown. It is impossible for the assigned compiler to work with two different languages. Therefore, in order to be able to work with multiple languages in your `\Code` folder, you must make some changes to the folder structure and to the `web.config` file.

The first step is to add two new subfolders to the `\Code` folder — a `\vb` folder and a `\cs` folder. This gives you the following folder structure:

```
\Code
    \VB
        Add.vb
    \CS
        Subtract.cs
```

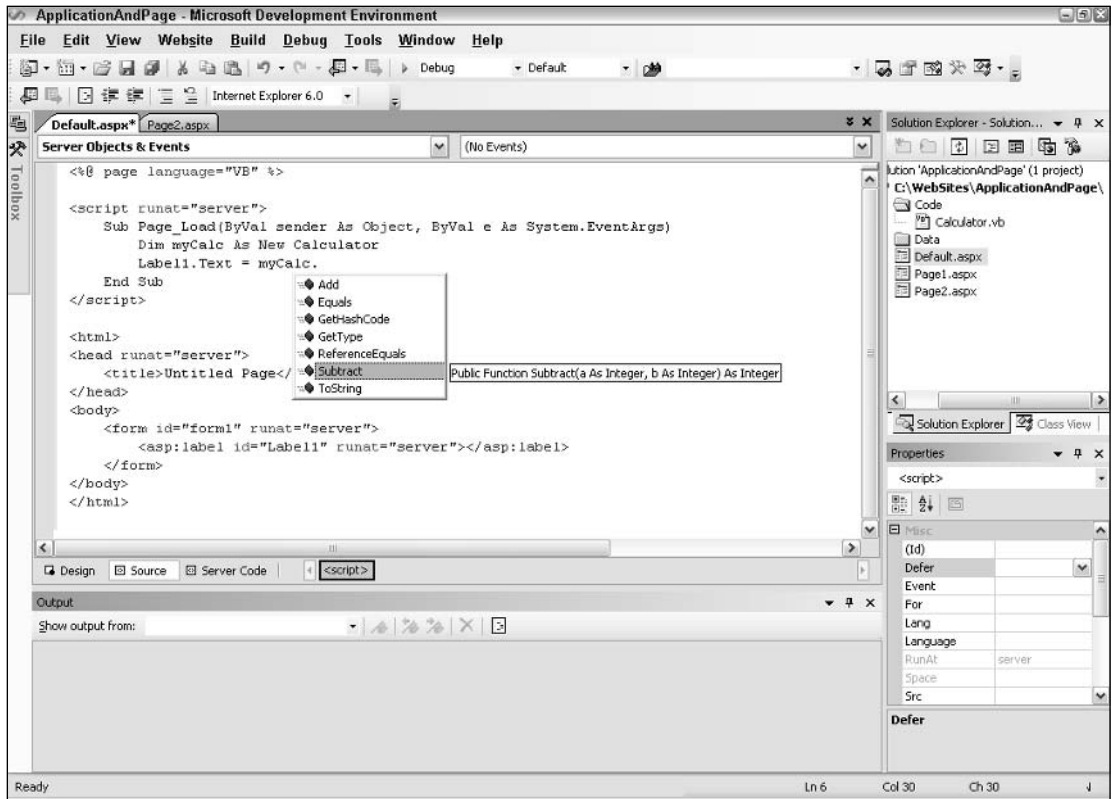


Figure 3-10

This still won't correctly compile these class files into separate assemblies, not until you make some additions to the `web.config` file. Most likely, you don't have a `web.config` file in your solution at this moment, so add one through the Solution Explorer. After it is added, change the `<compilation>` node so that it is structured as shown in Listing 3-16.

Listing 3-16: Fixing the `web.config` file so that you can have classes made up of different languages in the `\Code` folder

```
<compilation>
  <codeSubDirectories>
    <add directoryName="VB"></add>
    <add directoryName="CS"></add>
  </codeSubDirectories>
</compilation>
```

Now that this is in place in your `web.config` file, you can work with each of the classes in your ASP.NET pages. Also, any C# class placed in the `CS` folder is now automatically compiled just like any of the classes placed in the `VB` folder. It is also important to note that because of the ability to add these directories yourself in the `web.config` file, you are not required to name them `VB` and `CS` as I did; you can use whatever name tickles your fancy.

\Themes folder

Themes are a new way of providing a common look and feel to your site across every page. You implement a theme by using `.skin` file, CSS files, and images used by the server controls of your site. All these elements can make a Theme, which is then stored in the `\Themes` folder of your solution. By storing these elements within the `\Themes` folder, you ensure that all the pages within the solution can take advantage of the theme and easily apply its elements to the controls and markup of the page. Themes are discussed in great detail in Chapter 7 of this book.

\Resources folder

Resource files are string tables that can serve as data dictionaries for your applications when these applications require changes to content based on things such as changes in culture. You can add Assembly Resource Files (`.resx`) to this folder, and they are dynamically compiled and made part of the solution for use by your `.aspx` pages. When using ASP.NET 1.0/1.1, you were required to use the `resgen.exe` tool and to compile your resource files to a DLL or EXE for use within your solution. Now it is considerably easier to deal with resource files in ASP.NET 2.0.

In addition to strings, you can also add images and other files to your resource files. For an example of how to use resource files to create a multilingual ASP.NET 2.0 application, first create the `\Resources` folder in your application. For this example, create two resource files in this folder — `Resource.resx` and `Resource.fi-FI.resx`. The first file, `Resource.resx` is the default language file using American English. The second file is for the same text, but in the Finnish language. Hence, this file uses `fi-FI` in its name. When someone with a browser culture of `fi-FI` invokes the page, he sees the information that comes from this file (`Resource.fi-FI.resx`). Everyone else who comes to the site gets the information that comes from the other file (`Resource.resx`).

Notice (as shown in Figure 3-11) that you can actually do a lot with `.resx` files. The idea is to create a table of the items that need to be localized (such as text, images, and files). For this example, you can stick to text.

The `Resource.resx` file should have the following structure:

Name	Value
Answer	Hello there
PageTitle	Sample Page
Question	What is your name?

For the `Resource.fi-FI.resx` file, you should use the following structure:

Name	Value
Answer	Hei
PageTitle	Näytesivu
Question	Mikä sinun nimi on?

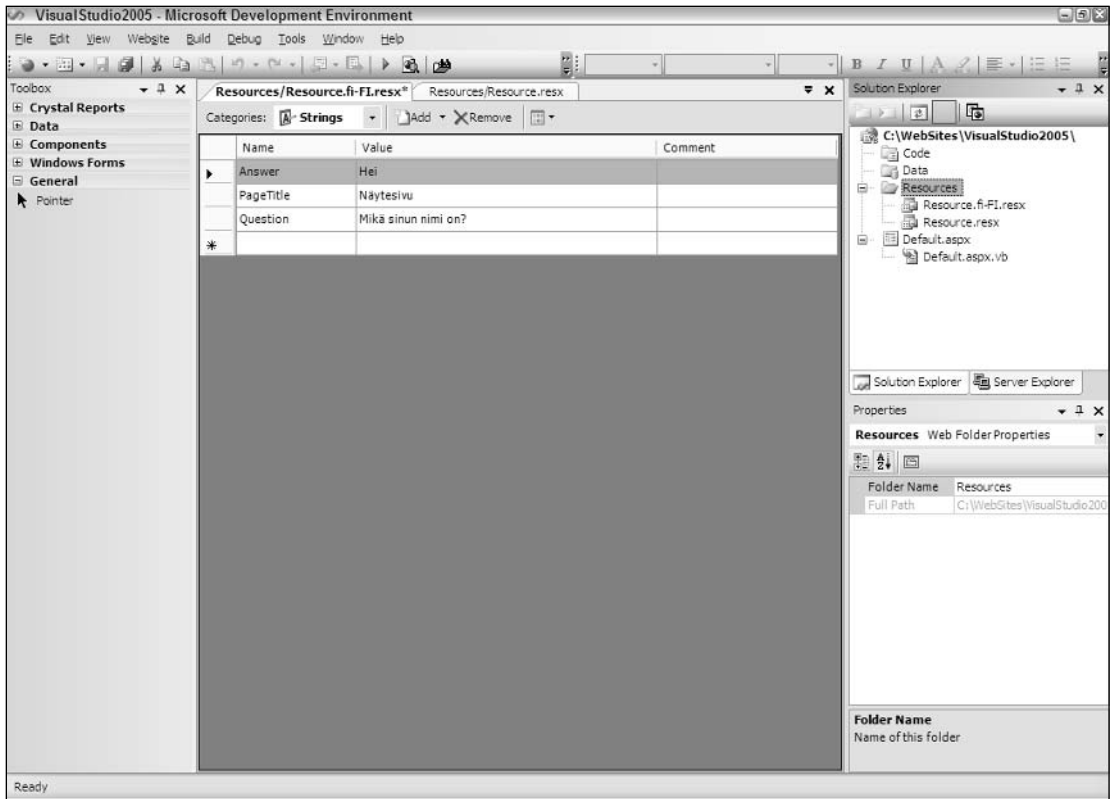


Figure 3-11

To use these files, create a simple .aspx page using the code from Listing 3-16.

Listing 3-16: A simple ASP.NET page that uses resource files

VB

```
<%@ Page Language="VB" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Page.Title = Resources.Resource.PageTitle
    End Sub

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = Resources.Resource.Answer & " " & Textbox1.Text
    End Sub
</script>
```

(continued)

Listing 3-16: (continued)

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title></title>
</head>
<body>
  <form id="Form1" runat="server">
    <p><%= Resources.Resource.Question %></p><br />
    <asp:textbox id="Textbox1" runat="server"></asp:textbox><br />
    <asp:button id="Button1" runat="server" text="Submit"
      onclick="Button1_Click" />
    <p><asp:label id="Label1" runat="server"></asp:label></p>
  </form>
</body>
</html>
```

C#

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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<script runat="server">
  void Page_Load(object sender, System.EventArgs e)
  {
    Page.Title = Resources.Resource.PageTitle;
  }

  void Button1_Click(object sender, System.EventArgs e)
  {
    Label1.Text = Resources.Resource.Answer + " " + Textbox1.Text;
  }
</script>
```

When this is run, you get the appropriate text based upon the culture setting in your browser. If this setting is not fi-FI, you get the American English text. The page output is shown in Figure 3-12.

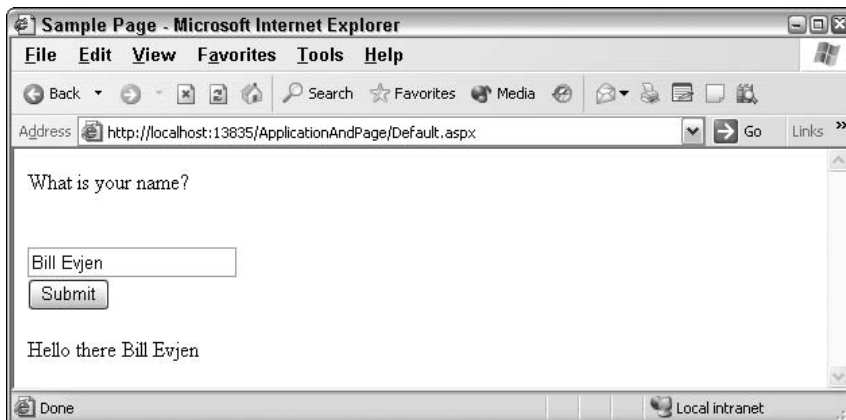


Figure 3-12

In order to see the Finnish text, add the following Page directive:

```
uiculture="fi-FI"
```

After this is in place, run the page. You see the Finnish language output shown in Figure 3-13.



Figure 3-13

Compilation

You already saw how Visual Studio 2005 compiles pieces of your application as you work with them (for instance, by placing a class in the `\Code` folder). The rest of the application, such as the `.aspx` pages themselves can be compiled just as in ASP.NET 1.0/1.1 by referencing the pages in the browser.

When an ASP.NET page is referenced in the browser for the first time, the request is passed to the ASP.NET parser that creates the class file in the language of the page. After the class file has been created, the class file is compiled into a DLL and then written to the disk of the Web server. This is detailed in Figure 3-14.

On the next request, great things happen. Instead of going through the entire process again for the second and respective requests, the request simply causes an instantiation of the already-created DLL, which sends out a response to the requester. This is illustrated in Figure 3-15.

Because of the mechanics of this process, if you made changes to your `.aspx` code-behind pages, you found it necessary to recompile your application. This can be quite a pain if you have a larger site and don't want your end users to experience the extreme lag that occurs when an `.aspx` page is referenced for the first time after compilation. Many developers, consequently, began to develop their own tools that would automatically go out and hit every single page within their application to remove this first-time lag hit from the end user's browsing experience.

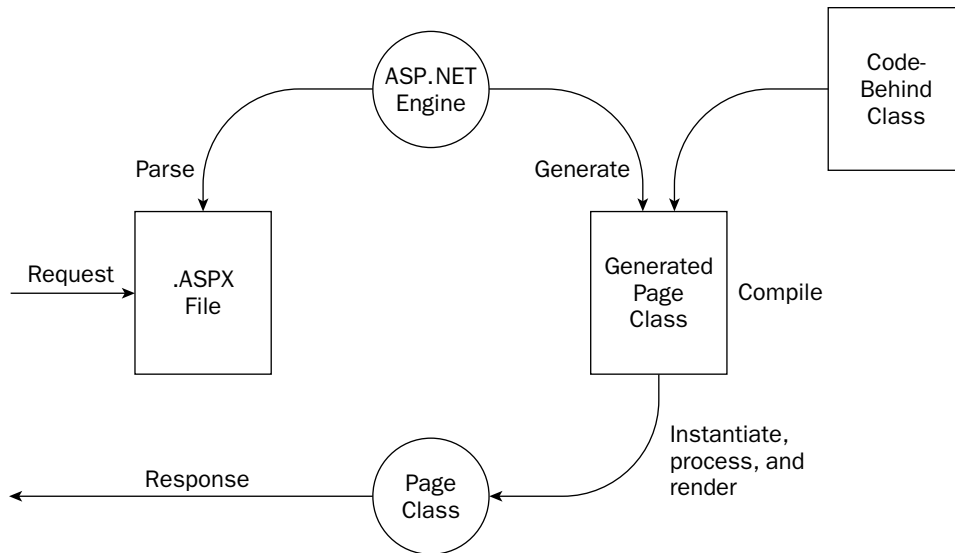


Figure 3-14

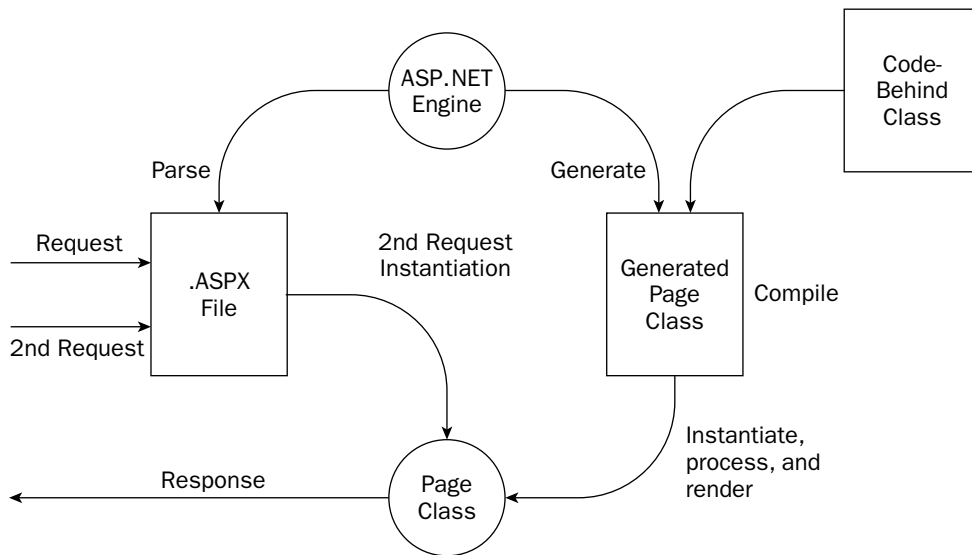


Figure 3-15

ASP.NET 2.0 introduces the technology to precompile your entire application with a single command that you can issue directly in the browser. This type of compilation is referred to as *in-place precompilation*. In order to precompile your entire ASP.NET application, pull up one of the pages in the browser and replace the page name with `precompile.axd`. So, if you are working with the Web server that is built into Visual Studio 2005, your request is structured in the following format:

```
http://[host]:[port]/[Application Name]/precompile.axd
```

If you are using IIS as the Web server, your request is structured in the following format:

```
http://[host]/[Application Name]/precompile.axd
```

If it is successful, you get a message that states the precompilation was successful. The other great thing about this precompilation capability is that you can also use it to find any errors on any of the ASP.NET pages in your application. Because it hits each and every page, if one of the pages contains an error that won't be triggered until runtime, you get notification of the error immediately as you invoke `precompile.axd`.

The next precompilation option is commonly referred to as *precompilation for deployment*. This is an outstanding new addition to ASP.NET that enables you to compile your application down to some DLLs, which can then be deployed to customers, partners, or elsewhere for your own use. Not only are minimal steps required to do this, but after your application is compiled, you only have to move around the DLL and some placeholder files for the site to work. This means that your Web site code is completely removed and placed in the DLL when deployed.

To precompile your application for deployment, you must use the `aspnet_compiler.exe` tool that now comes with ASP.NET 2.0. You navigate to the tool using the Command window. Open the Command window and navigate to `C:\Windows\Microsoft.NET\Framework\v2.0.xxxxx\`. When you are there, you can work with the `aspnet_compiler` tool.

Before you do, however, create a folder in your root drive called, for example, `Wrox`. This folder is the one you ask the compiler to output to. When it is in place, you can return to the compiler tool and give it the following command:

```
aspnet_compiler -v [Application Name] -p [Physical Location] [Target]
```

So, if you had an application called `INETA` located at `C:\Websites\INETA`, you would use the following commands:

```
aspnet_compiler -v /INETA -p C:\Websites\INETA C:\Wrox
```

Press Return and the compiler either tells you that it has a problem with one of the command parameters, or that it was successful (shown in Figure 3-16). If it was successful, you can see the output that was placed in the target directory.

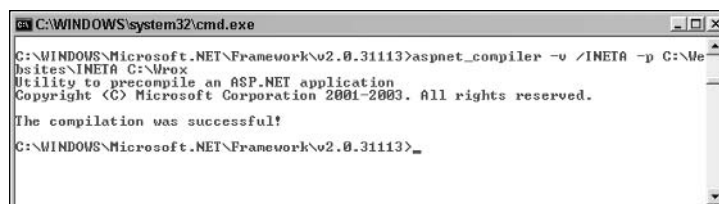


Figure 3-16

Chapter 3

In the example just shown, `-v` is a command for the virtual path of the application — which is provided by using `/INETA`. The next command is `-p`, which is pointing to the physical path of the application. In this case, it is `C:\Websites\INETA`. Finally, the last bit, `C:\Wrox`, is the location of the compiler output. The following table describes the possible commands for the `aspnet_compiler.exe` tool.

Command	Description
-m	Specifies the full IIS metabase path of the application. If you use the <code>-m</code> command, you cannot use the <code>-v</code> or <code>-p</code> commands.
-v	Specifies the virtual path of the application that is going to be compiled. If you also use the <code>-p</code> command, the physical path is used to find the location of the application.
-p	Specifies the physical path of the application that is going to be compiled. If this is not specified, the IIS metabase is used to find the application.
targetDir	Specifies the target directory where the compiled files should be placed. If this is not specified, the files output are placed in the application directory.

After compiling the application, you can go to `C:\Wrox` to see the output. Here you see all the files and the file structure that was in the original application. But if you look at the contents of one of the files, notice that the file is simply a placeholder. In the actual file you find the comment:

```
This is a marker file generated by the precompilation tool,  
and should not be deleted!
```

In fact, you find a `Code.dll` file in the `bin` folder where all the page code is located. Because it is in a DLL file, it provides great code obfuscation as well. From here on, all you do is move these files to another server using FTP or Windows Explorer and you can run the entire Web application from these files. When you have an update to the application, you simply provide a new set of compiled files. A sample output is displayed in Figure 3-17.

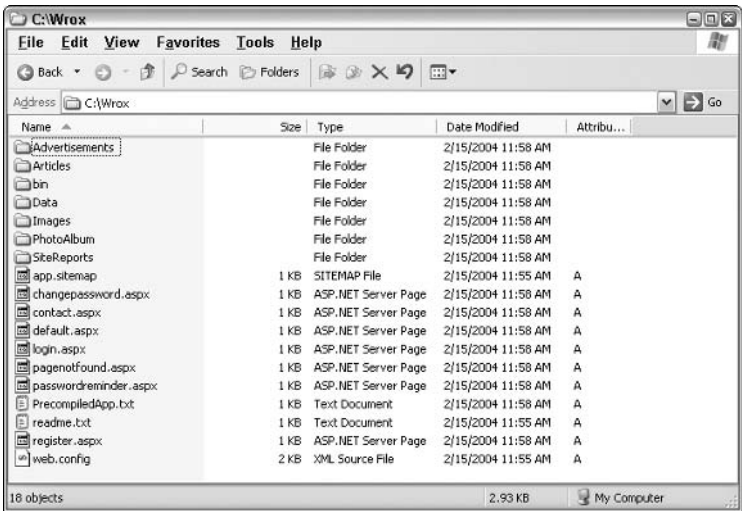


Figure 3-17

Note that this compilation process doesn't compile *every* type of Web file. In fact, it compiles only the ASP.NET-specific file types and leaves out of the compilation process files such as

- ☐ HTML files
- ☐ XML files
- ☐ XSD files
- ☐ `Web.Config` files
- ☐ Text files

You can't do much to get around this, except in the case of the HTML files and the text files. For these file types, just change the file extension to `.aspx` and they are then compiled into the `Code.dll` like all the other ASP.NET files.

Summary

This chapter covered a lot of ground. I discussed some of the issues concerning ASP.NET applications as a whole and the choices you have when building and deploying these new applications. With the help of Visual Studio 2005, you now have options about which Web server to use when building your application and whether to work locally or remotely through the new built-in FTP capabilities.

ASP.NET 2.0 and Visual Studio 2005 make it easy to build your pages using an inline coding model or to select a new and better code-behind model that is simpler to use and easier to deploy. You also took a look at the new cross-posting capabilities and the new fixed folders that ASP.NET 2.0 has incorporated to make your life easier. These folders make their resources available dynamically with no work on your part. Finally, you saw some of the outstanding new compilation options that you have at your disposal.

As you worked through some of the examples, you may have been thinking, "WOW!" But wait . . . there's plenty more to come!

4

New Ways to Handle Data

ASP.NET 1.0 introduced some revolutionary ways to retrieve and manipulate data. ADO.NET (introduced in .NET 1.0) enabled you to grab data from data stores in an intuitive manner and then store the retrieved data in objects like the new DataSet object. Although the process was revolutionary, it was complicated and contained numerous possible pitfalls.

ASP.NET 2.0 introduces data source controls to bridge the gap between your data stores and the data-bound controls at your disposal. These new data controls not only enable you to retrieve data from various data stores, but they also let you easily manipulate the data (using paging, sorting, editing, and filtering) before the data is bound to an ASP.NET server control.

This chapter presents the new data source controls as well as some of the new data-bound controls that you can use to display retrieved data.

The New Data Source Controls

Before the introduction of the new data source controls, just retrieving data from a data store and displaying it in a DataGrid control was a multistep process. This is illustrated in Listing 4-1.

Listing 4-1: Binding a DataGrid control in ASP.NET 1.0/1.1 (VB only)

```
Dim conn As SqlConnection = New SqlConnection("server='localhost';  
    trusted_connection=true; Database='Northwind'")  
Dim cmd As SqlCommand = New SqlCommand("Select * From Customers", conn)  
conn.Open()  
  
Dim da As SqlDataAdapter = New SqlDataAdapter(cmd)  
Dim ds As New DataSet  
  
da.Fill(ds, "Customers")  
  
DataGrid1.DataSource = ds  
DataGrid1.DataBind()
```

In this example, you can see the many steps required just to retrieve the Customers table from SQL Server. First, a `SqlConnection` object is created to connect to SQL Server. Next, a `SqlCommand` object is created to pass in the command to the database from which data is to be retrieved. After the connection is opened, `SqlDataAdapter` is used to get the data and then fill the `DataSet` object that is created. After the `DataSet` object is in place and filled with the data from the Customers table, the `DataSet` object is bound to the `DataGrid` control with a `DataBind()` command.

Because ASP.NET 2.0 is backward compatible, the example shown in Listing 4-1 works just the same as it did in earlier versions of ASP.NET. Now, however, you can use one of the six new data source controls that have been introduced in ASP.NET 2.0. These controls provide a declarative way to connect to data stores and retrieve specific data. Working with the new data source controls is considerably easier than any previous methods. In most cases, you simply need a single line of code to get at the data you want and, in some cases, zero lines of code are required.

The new data source controls include some specifically designed to work with Microsoft SQL Server, Microsoft Access, and many other types of data stores. The following table details the new data source controls available in ASP.NET 2.0.

Data Source Control	Description
<code>SqlDataSource</code>	Enables you to work with any SQL-based database, such as Microsoft SQL Server or Oracle.
<code>AccessDataSource</code>	Enables you to work with a Microsoft Access file (.mdb).
<code>ObjectDataSource</code>	Enables you to work with a business object or a Visual Studio 2005 data component.
<code>XmlDataSource</code>	Enables you to work with the information from an XML file or an XML source (for example an RSS feed).
<code>SiteMapDataSource</code>	Enables you to work with the hierarchical data represented in the site map file (.sitemap). These files and how to bind to them are discussed in Chapter 5.
<code>DataSetDataSource</code>	Enables you to work with data that is represented in a <code>DataSet</code> object.

These data source controls connect to the assigned data store, retrieve the data, and perform any manipulations on the data that you specify using control attributes. The data source control does all the sorting, paging, and editing of the data. It also works with any data-bound controls such as the new `GridView` control to perform automatic databinding without any work on your part.

The Data-Bound Server Controls

ASP.NET 2.0 provides a large collection of new data-bound server controls that can be used in conjunction with the new data source controls to display retrieved data in the browser. Although you probably recognize many controls from ASP.NET 1.0/1.1, you also meet some new server controls — such as the

GridView control. Throughout this chapter, I show you how to use data-bound server controls with these new data source controls. Remember the principles I describe can be easily applied to the other data-bound controls as well.

The data-bound controls in ASP.NET 2.0 include

- | | |
|--|--|
| <input type="checkbox"/> <asp:GridView> | <input type="checkbox"/> <asp:DropDownList> |
| <input type="checkbox"/> <asp:DataGrid> | <input type="checkbox"/> <asp:BulletedList> |
| <input type="checkbox"/> <asp:DetailsView> | <input type="checkbox"/> <asp:CheckBoxList> |
| <input type="checkbox"/> <asp:TreeView> | <input type="checkbox"/> <asp:RadioButtonList> |
| <input type="checkbox"/> <asp:Menu> | <input type="checkbox"/> <asp:ListBox> |
| <input type="checkbox"/> <asp:DataList> | <input type="checkbox"/> <asp:AdRotator> |
| <input type="checkbox"/> <asp:Repeater> | |

The next section shows you how to use some of these controls with the new data source controls.

The SqlDataSource and GridView Controls

The SqlDataSource control can be used with one of the newest data-bound controls in ASP.NET 2.0 — the GridView control. The SqlDataSource control is not only for use with Microsoft's SQL Server, you can also use it for any SQL-capable server. For example, you can use this data source control to connect to any OleDb- or ODBC-based data store or you can use it to work with Oracle.

Reading data

You can use this data source control to read data from a Microsoft SQL Server database. In this case, you can use the sample Northwind database that comes with SQL Server. To start, place a new GridView control on your page. This is the new tabular control that replaces the DataGrid control from ASP.NET 1.0/1.1. I discuss this control in detail throughout this chapter.

After the GridView is in place, change the look and feel of the control by clicking the Auto Format link in the smart tag and choosing Black & Blue 1. Next, drag and drop a SqlDataSource control onto the design surface. Because this control is not visual, it appears as a simple gray box on the design surface. Go to the page source code by clicking the Source tab in Visual Web Developer. Modify the SqlDataSource control so that it appears as shown in Listing 4-2.

Listing 4-2: Reading data from a SqlDataSource control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SqlDataSource Control Page</title>
</head>
```

(continued)

Listing 4-2: (continued)

```
<body>
  <form id="form1" runat="server">
    <asp:GridView ID="GridView1" Runat="server" DataSourceId="SqlDataSource1"
      BackColor="White" GridLines="Vertical"
      BorderStyle="Solid" Cellpadding="3" ForeColor="Black"
      BorderColor="#999999" BorderWidth="1px">
      <FooterStyle BackColor="#CCCCCC">
      </FooterStyle>
      <SelectedRowStyle ForeColor="White" BackColor="#000099"
        Font-Bold="True">
      </SelectedRowStyle>
      <PagerStyle ForeColor="Black" HorizontalAlign="Center"
        BackColor="#999999">
      </PagerStyle>
      <HeaderStyle ForeColor="White" BackColor="Black" Font-Bold="True">
      </HeaderStyle>
      <AlternatingRowStyle BackColor="#CCCCCC">
      </AlternatingRowStyle>
    </asp:GridView>
    <br />
    <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
      SelectCommand="Select * From Customers"
      ProviderName="System.Data.OleDb"
      ConnectionString="Provider=SQLOLEDB;Server=localhost;uid=sa;pwd=password;
        database=Northwind" />
  </form>
</body>
</html>
```

In this example, you can see that you need only a couple of attributes in the `SqlDataSource` control to read data from SQL Server. The first is the `SelectCommand` attribute. This takes a `String` value that is the SQL command you want the `SqlDataSource` control to use. In this example, you simply select everything that is in the `Customers` table of the `Northwind` database. The next important attribute is the `ConnectionString` attribute. This attribute takes a `String` value. The connection string must be the kind of string that the provider expects, which is defined in the `ProviderName` attribute. In this case, it is the `OleDb` provider.

After your `SqlDataSource` control is in place, you can attach any of the data-bound controls to this data source control. To do this, the `GridView` control on the page uses the `DataSourceId` attribute. The `DataSourceId` attribute takes a `String` value of the control ID of the `SqlDataSource` control — `SqlDataSource1`.

When this table is generated in the browser, you see all the information from the `Customers` table in the `Northwind` database, as shown in Figure 4-1.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745
BSBEV	B's Beverages	Victoria Ashworth	Sales Representative	Fauntleroy Circus	London		EC2 5NT	UK	(171) 555-1212	
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892

Figure 4-1

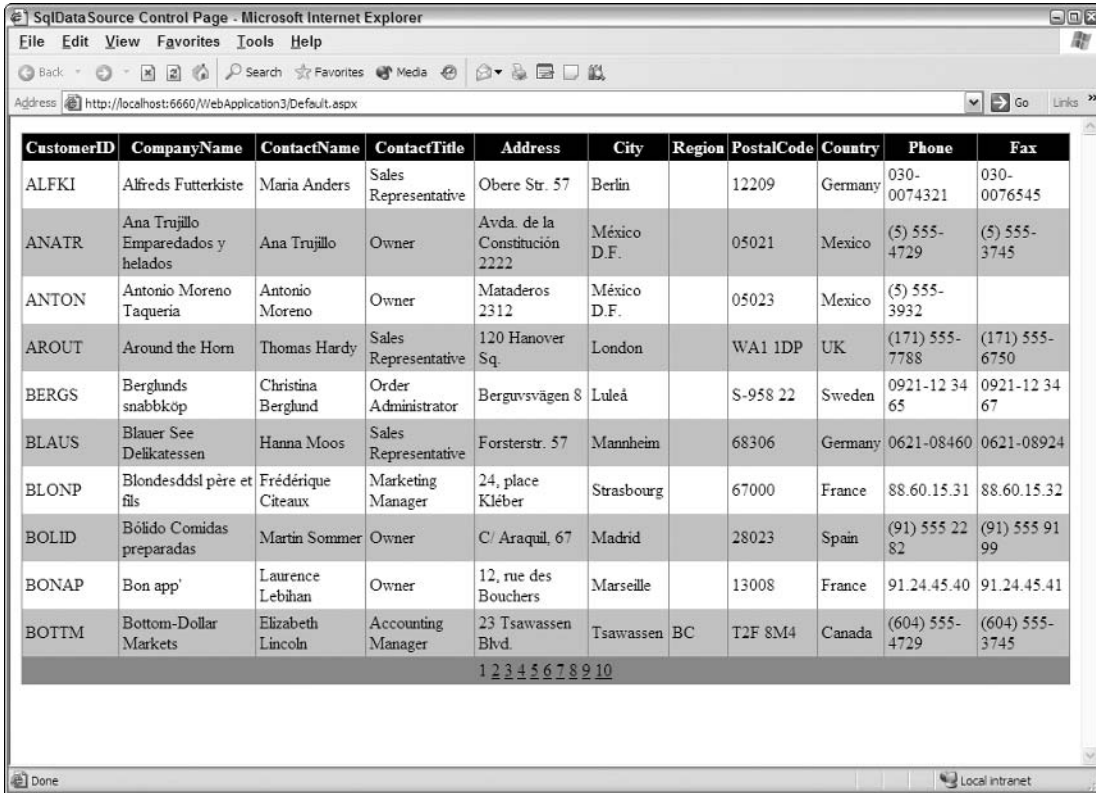
Applying paging in the GridView

Another formerly cumbersome task for the DataGrid control was paging. With ASP.NET 2.0, the new GridView control can easily work with any of the data source controls to apply paging to the data you are working with. To apply paging to the GridView, simply use the `AllowPaging` attribute set to `True`. By default, its value is `False`. The use of the `AllowPaging` attribute is shown in Listing 4-3.

Listing 4-3: Turning on paging in the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceId="SqlDataSource1"
  BackColor="White" GridLines="Vertical" AllowPaging="True"
  BorderStyle="Solid" CellPadding="3" ForeColor="Black"
  BorderColor="#999999" BorderWidth="1px">
  ...
  <PagerStyle ForeColor="Black" HorizontalAlign="Center" BackColor="#999999">
  </PagerStyle>
  ...
</asp:GridView>
```

It really is as simple as that. You don't need to make any changes to the `SqlDataSource` control for this to work. After the `AllowPaging` attribute is changed to `True`, the GridView appears as shown in Figure 4-2.



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745

1 2 3 4 5 6 7 8 9 10

Figure 4-2

In the figure, you can also see that the numbers at the bottom of the page have a style applied to them. This style is defined in the GridView control with the use of the `<PagerStyle>` element. Besides the `<PagerStyle>` element, a number of additional style elements can be utilized within the GridView control.

As you can see from the screen shot in Figure 4-2, the page numbers are shown at the bottom of the table. You can completely customize the appearance of these numbers. You can, for example, use shortcuts to change the navigation among pages in the table.

In IntelliSense, you see a collection of attributes that start with the term `PagerSettings-`. The `PagerSettings-mode` attribute, for example, can take `NextPrevious`, `NextPreviousFirstLast`, `Numeric`, or `NumericFirstLast` as possible values. The default value is `Numeric`.

Figure 4-3 shows a GridView control using `NextPrevious`.

	Sales Agent	35 King George	London		WX3
el	Sales Manager	Kirchgasse 6	Graz		8010
< >					

Figure 4-3

Figure 4-4 shows the use of NextPreviousFirstLast.

	Sales Agent	35 King George	London		WX3
el	Sales Manager	Kirchgasse 6	Graz		8010
<< < > >>					

Figure 4-4

The PagerSettings-PageButtonCount is an interesting attribute that can be used with the two numeric settings for the PagerSettings-Mode attribute (one of which is the default setting). This attribute takes an Integer value and, when this value is set, only the specified number of page options in the list of pages is shown. Listing 4-4 is an example of this.

Listing 4-4: Showing only a specific number of page options

```
<asp:GridView ID="GridView1" Runat="server" DataSourceId="SqlDataSource1"
  BackColor="White" GridLines="Vertical" AllowPaging="True"
  PagerSettings-Mode="Numeric" PagerSettings-PageButtonCount="3"
  BorderStyle="Solid" CellPadding="3" ForeColor="Black"
  BorderColor="#999999" BorderWidth="1px">
  ...
  <PagerStyle ForeColor="Black" HorizontalAlign="Center" BackColor="#999999">
  </PagerStyle>
  ...
</asp:GridView>
```

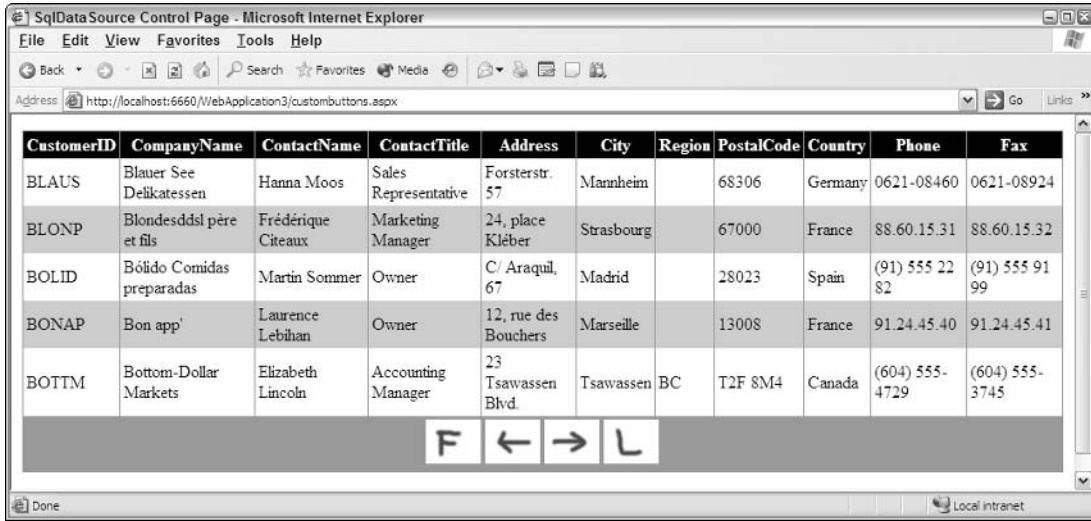
Using these attribute combinations causes the pager at the bottom of the table to appear as shown in Figure 4-5.

Sales Associate	Maubelstr. 90	Brandenburg		1477
Sales Representative	67, avenue de l'Europe	Versailles		7800
... 4 5 6 ...				

Figure 4-5

Chapter 4

I won't go through examples of the other attributes and how they relate to the paging capabilities of the GridView. Some of these cool attributes, however, include the `PagerSettings-FirstPageImageUrl`, `PagerSettings-FirstPageText`, `PagerSettings-LastPageImageUrl`, and the `PagerSettings-LastPageText` attributes. When these are used with the `PagerSettings-mode` attribute set to one of the `NextPrevious` styles, you can have images or custom text appear in place of the greater-than/less-than signs, as illustrated in Figure 4-6.



The screenshot shows a Microsoft Internet Explorer window displaying a web application. The address bar shows `http://localhost:6660/WebApplication3/custombuttons.aspx`. The main content area displays a GridView control with a table of customer data. The table has 11 columns: CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, and Fax. The data is as follows:

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen	BC	T2F 8M4	Canada	(604) 555-4729	(604) 555-3745

Below the table, there is a custom pager with four buttons: 'F', '<', '>', and 'L'.

Figure 4-6

Sorting rows in the GridView control

Using the same GridView control from the previous examples, you can enable the end user to sort rows as easily as he can paginate them. This is quite useful when dealing with a large collection of data in the GridView. If you can sort the data by clicking the column heading you can easily view the data in a logical manner. How to apply sorting to the data in the GridView control is shown in Listing 4-5.

Listing 4-5: Turning on sorting in the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceId="SqlDataSource1"
  BackColor="White" GridLines="Vertical" AllowSorting="True"
  BorderStyle="Solid" CellPadding="3" ForeColor="Black"
  BorderColor="#999999" BorderWidth="1px">
  ...
</asp:GridView>
```

If you set the value of the `AllowSorting` attribute to `True`, the end user can click a column heading to sort the entire list based on that selected column. Clicking the same column again sorts the list in the reverse direction. Figure 4-7 shows the GridView sorted by `Country`.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
CACTU	Cactus Comidas para llevar	Patricio Simpson	Sales Agent	Cerrito 333	Buenos Aires		1010	Argentina	(1) 135-5555	(1) 135-4892
OCEAN	Océano Atlántico Ltda.	Yvonne Moncada	Sales Agent	Ing. Gustavo Moncada 8585 Piso 20-A	Buenos Aires		1010	Argentina	(1) 135-5333	(1) 135-5535
RANCH	Rancho grande	Sergio Gutiérrez	Sales Representative	Av. del Libertador 900	Buenos Aires		1010	Argentina	(1) 123-5555	(1) 123-5556
ERNSH	Ernst Handel	Roland Mendel	Sales Manager	Kirchgasse 6	Graz		8010	Austria	7675-3425	7675-3426
PICCO	Piccolo und mehr	Georg Pippis	Sales Manager	Geistweg 14	Salzburg		5020	Austria	6562-9722	6562-9723
MAISD	Maison Dewey	Catherine Dewey	Sales Agent	Rue Joseph-Bens 532	Bruxelles		B-1180	Belgium	(02) 201 24 67	(02) 201 2-68

Figure 4-7

Defining bound columns in the GridView control

In many instances, you want to display only select columns from the table of data being retrieved. One way to do this is to modify your `select` statement so that it retrieves only the columns you want to display. For instance, your `Select` statement might resemble the following:

```
Select CustomerId, CompanyName From Customers
```

This code determines that only these two columns are displayed in the GridView control. At times, you may want to retrieve certain columns in order to act upon the information they contain, but not display these columns to the end user. For example, you may want to retrieve the `CustomerId` column from the Customers table, but not display this column to the end user. You see how you can do this in Listing 4-6.

Listing 4-6: Building columns in the GridView control

```
<asp:GridView ID="GridView1" Runat="server" DataSourceId="SqlDataSource1"
  AllowSorting="True" BackColor="White" GridLines="Vertical"
  BorderStyle="Solid" CellPadding="3" ForeColor="Black" BorderColor="#999999"
  BorderWidth="1px" AutoGenerateColumns="False">
  <Columns>
    <asp:BoundField Visible="False" DataField="CustomerId">
    </asp:BoundField>
    <asp:BoundField SortExpression="CompanyName" HeaderText="Company Name"
      DataField="CompanyName">
    </asp:BoundField>
    <asp:BoundField SortExpression="Country" HeaderText="Country"
      DataField="Country">
    </asp:BoundField>
  </Columns>
  ...
</asp:GridView>
```

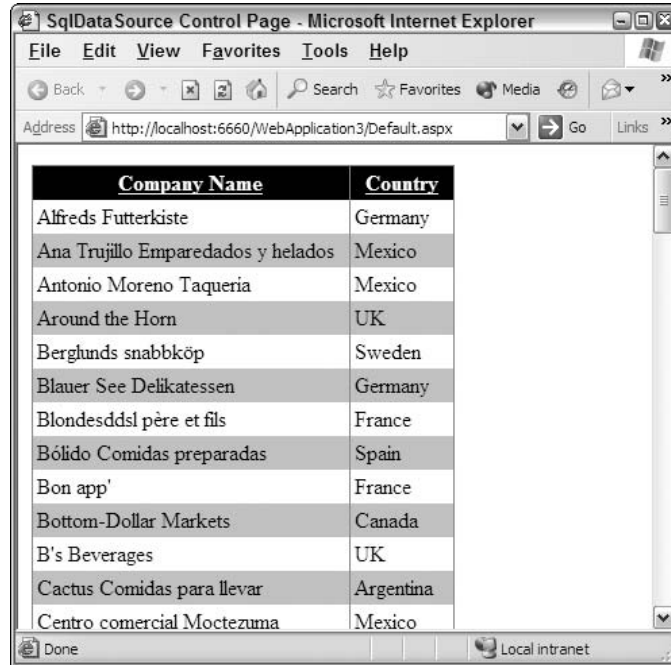
For the example in Listing 4-6, I changed the SQL statement in the `SqlDataSource` control to `Select * From Customers` just to show that you can pick and choose which items to display in the `GridView` control. The attribute `AutoGenerateColumns` is added to the main `<asp:GridView>` node and set to `False`. By default, this attribute is set to `True` causing the `GridView` control to automatically generate each and every column that comes from the dataset retrieved by the `SqlDataSource` control. When the attribute is set to `False`, you must specify which columns the `GridView` control generates.

You can also use the `<Columns>` section to specify how the columns are displayed in the `GridView` control. The `< Columns >` section of the `GridView` control can take any number of `<asp:BoundField>` elements. The `<asp:BoundField>` elements are what you use to associate a column displayed in the `GridView` control to a column of data from the `Customers` table in the `Northwind` database. You can see that each of the three `<asp:BoundField>` elements in this `GridView` control example is used in a different manner. The first `<asp:BoundField>` element grabs the `CustomerId` field. This is specified in the `<asp:BoundField >` element by the `DataField` attribute. The `String` value assigned to the `DataField` element must be the column name used in the `Customers` table. The other important attribute used in the `<asp:BoundField>` is the `Visible` attribute. By default, the `Visible` attribute is set to `True` — meaning that the column is displayed in the `GridView` control; but in this case, you set it to `False`. Why? Because when the end user is acting on a row of data, you may want to associate that row of data with an identifier (such as a primary key value). You may not, however, want to show that data item to the end user. This is illustrated later in the chapter in the example where users edit rows of data.

The second `<asp:BoundField>` element is associated with the `CompanyName` column in the `Customers` table via the use of the `DataField` attribute. This `<asp:BoundField>` element also has a few additional items contained within it. The first is the `HeaderText` attribute. Column names contained with databases are not the prettiest things in the world, nor are they always that descriptive. For this reason, you often change the column header names in the `GridView` control to make them a little more user-friendly. You make this change using the `HeaderText` attribute. The `String` value placed here is used by the `GridView` control at the top of the column. In this case, you change the default display of `CompanyName` to `Company Name`. The second attribute used in the second `<asp:BoundField>` element is the `SortExpression` attribute. This attribute points to the column name used in the `Customers` table. When sorting is enabled, the `GridView` knows on which column to sort the data.

In the third `<asp:BoundField>`, you don't use the `HeaderText` attribute because `Country` is a good heading listing. The result of Listing 4-6 is illustrated in Figure 4-8.

When you use the `<asp:BoundField>` elements, the columns appear from left to right in the order in which they are used in the `<Columns>` section of the `GridView` control. Therefore, you can easily change the order in which they appear in the `GridView` control just by changing their order in the `<Columns>` section.



The screenshot shows a Microsoft Internet Explorer window titled "SqlDataSource Control Page - Microsoft Internet Explorer". The address bar displays "http://localhost:6660/WebApplication3/Default.aspx". The main content area contains a table with two columns: "Company Name" and "Country". The table lists various companies and their respective countries, alternating row colors for readability. The status bar at the bottom shows "Done" and "Local intranet".

Company Name	Country
Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taqueria	Mexico
Around the Horn	UK
Berglunds snabbköp	Sweden
Blauer See Delikatessen	Germany
Blondesddsl père et fils	France
Bólido Comidas preparadas	Spain
Bon app'	France
Bottom-Dollar Markets	Canada
B's Beverages	UK
Cactus Comidas para llevar	Argentina
Centro comercial Moctezuma	Mexico

Figure 4-8

Another cool feature of the GridView control is that you can use images in the headings instead of text. This is shown in Listing 4-7.

Listing 4-7: Using images in the column headings

```
<asp:BoundField SortExpression="Country" DataField="Country"
  HeaderImageUrl="~/iconGlobe.gif">
</asp:BoundField>
```

This is a partial code listing. The `<asp:BoundField>` element associated with the Country heading is changed so that it now displays an image instead of text. The `HeaderText` attribute and value is removed and replaced with the `HeaderImageUrl` attribute. The value for this attribute points to the location of the image. This kind of construct gives you something similar to Figure 4-9.

Company Name	
Alfreds Futterkiste	Germany
Ana Trujillo Emparedados y helados	Mexico
Antonio Moreno Taqueria	Mexico
Around the Horn	UK
Berglunds snabbköp	Sweden
Blauer See Delikatessen	Germany
Blondesdddsl père et fils	France
Bóldo Comidas preparadas	Spain
Bon app'	France
Bottom-Dollar Markets	Canada
B's Beverages	UK
Cactus Comidas para llevar	Argentina
Centro comercial Moctezuma	Mexico
Cho-p-suev Chinese	Switzerland

Figure 4-9

Another nice feature is that end users can click the image to sort the column (just as they clicked text column headers). An interesting feature associated with column generation is the capability to specify what the GridView should display when it encounters `Null` values within a column. For an example of this, add a column using an additional `<asp:BoundField>`, as illustrated in Listing 4-8.

Listing 4-8: Working with Null values in a column

```
<asp:BoundField SortExpression="Region" NullDisplayText="NO REGION"
  HeaderText="Region" DataField="Region">
</asp:BoundField>
```

For this example, add an `<asp:BoundField>` element to display the `Region` column from the `Customers` table. As you look through the data in the `Region` section, notice that not every row has a value in it. What if you didn't want just a blank box to show an empty value, but you wanted to show some text in place of the empty items in the column. To do this, you utilize the `NullDisplayText` attribute. The `String` value it provides is used for each and every row that doesn't have a `Region` value. This construct produces the results illustrated in Figure 4-10.

Company Name	Country	Region
Alfreds Futterkiste	Germany	NO REGION
Ana Trujillo Emparedados y helados	Mexico	NO REGION
Antonio Moreno Taqueria	Mexico	NO REGION
Around the Horn	UK	NO REGION
Berglunds snabbköp	Sweden	NO REGION
Blauer See Delikatessen	Germany	NO REGION
Blondesddsl père et fils	France	NO REGION
Bólido Comidas preparadas	Spain	NO REGION
Bon app'	France	NO REGION
Bottom-Dollar Markets	Canada	BC
B's Beverages	UK	NO REGION
Cactus Comidas para llevar	Argentina	NO REGION
Centro comercial Moctezuma	Mexico	NO REGION
Chop-suey Chinese	Switzerland	NO REGION
Comércio Mineiro	Brazil	SP

Figure 4-10

A `Null` value is different from an empty string value. In the example from Listing 4-8, any empty strings encountered do not receive the `NO REGION` text. To get around this (if you want to), you can use the `TreatEmptyStringAsNull` attribute, which takes a `Boolean` value. If set to `True`, all empty values are treated as if they are `Null` values — meaning that they receive the `NO REGION` text. If the attribute is set to `False` (the default), all empty values are treated as something other than `Null` and do not receive the `NO REGION` text.

Enabling the editing of rows in the GridView control

Not only do developers want to display tabular data within a browser, they also want to give end users the capability to edit and send the changes made back to the data store. Adding an editing capability to the `DataGrid` control in ASP.NET 1.0/1.1 was always difficult. But it was important enough that developers often felt the need to add it to their pages.

ASP.NET 2.0's new `GridView` server control allows for easy editing of the content it contains. For an example of this, enable the end user to edit the contents contained in the `GridView` control, as shown in Listing 4-9.

Listing 4-9: Editing data in the GridView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SqlDataSource Control Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" Runat="server" Datasourceid="SqlDataSource1"
            AutoGenerateColumns="False" DataKeyNames="CustomerID"
            BackColor="White" GridLines="Vertical"
            BorderStyle="Solid" CellPadding="3" ForeColor="Black"
            BorderColor="#999999" BorderWidth="1px">
            <Columns>
                <asp:CommandField ShowEditButton="True">
                </asp:CommandField>
                <asp:BoundField SortExpression="CustomerID" HeaderText="CustomerID"
                    ReadOnly="True" DataField="CustomerID">
                </asp:BoundField>
                <asp:BoundField SortExpression="CompanyName"
                    HeaderText="Company Name" DataField="CompanyName">
                </asp:BoundField>
                <asp:BoundField SortExpression="Country" HeaderText="Country"
                    DataField="Country">
                </asp:BoundField>
                <asp:BoundField SortExpression="Region" NullDisplayText="NO REGION"
                    HeaderText="Region" DataField="Region">
                </asp:BoundField>
            </Columns>
            <FooterStyle BackColor="#CCCCCC">
            </FooterStyle>
            <SelectedRowStyle ForeColor="White" BackColor="#000099"
                Font-Bold="True">
            </SelectedRowStyle>
            <PagerStyle ForeColor="Black" HorizontalAlign="Center"
                BackColor="#999999">
            </PagerStyle>
            <HeaderStyle ForeColor="White" BackColor="Black" Font-Bold="True">
            </HeaderStyle>
            <AlternatingRowStyle BackColor="#CCCCCC">
            </AlternatingRowStyle>
        </asp:GridView>
        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
            SelectCommand="Select * From Customers"
            UpdateCommand="UPDATE Customers SET CompanyName = @CompanyName,
                Country = @Country, Region = @Region WHERE (CustomerID = @CustomerID)"
            ConnectionString="Server=(local);Trusted_Connection=True;Integrated
                Security=SSPI;Persist Security Info=True;Database=Northwind"
            ProviderName="System.Data.SqlClient">
            <UpdateParameters>
                <asp:Parameter Name="CustomerID" Type="String">
```

```

        </asp:Parameter>
        <asp:Parameter Name="CompanyName" Type="String">
        </asp:parameter>
        <asp:parameter Name="Country" Type="String">
        </asp:parameter>
        <asp:parameter Name="Region" Type="String">
        </asp:parameter>
        </UpdateParameters>
    </asp:SqlDataSource>
</form>
</body>
</html>

```

When you run the table produced by the GridView, it displays the contents as specified in the earlier examples in this chapter. The new addition is the Edit column that appears in the leftmost column in the table. Clicking any Edit link enables the end user to edit the content of the selected row. This is illustrated in Figure 4-11.

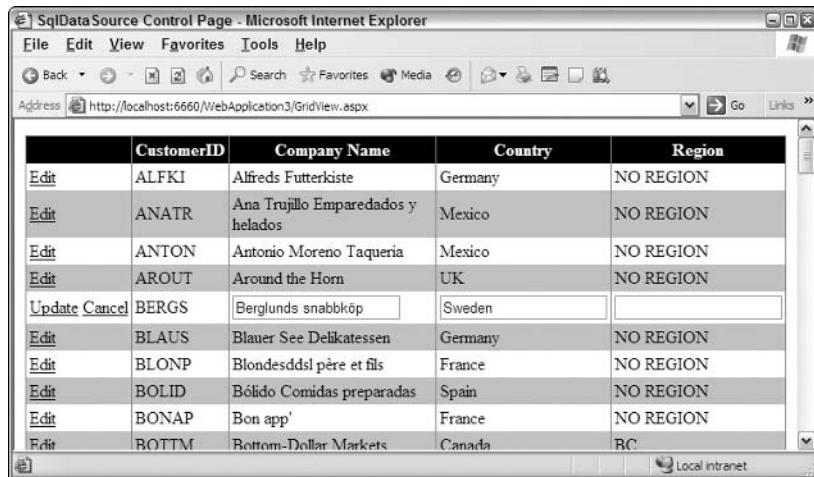


Figure 4-11

Looking at the code in this example, note that you have picked the columns from the Customers table to be displayed. In this case, the columns displayed are the CustomerID, CompanyName, Country, and Region columns. In the listings contained within the <Columns> section of the GridView, notice that the leftmost column in the GridView is the Edit link, which enables the end user to edit a selected row:

```

<asp:CommandField ShowEditButton="True">
</asp:CommandField>

```

To create the Edit column in the table, you use the <asp:CommandField> element, which enables you to place actions like Cancel, Delete, Edit, Insert, and Select as buttons in your tables. In this case, you show the Edit button by setting the ShowEditButton attribute to True. Just like the <asp:BoundField> element, the <asp:CommandField> element can be modified with a large number of different style attributes, as well as with attributes such as the HeaderText and the HeaderImageUrl.

Chapter 4

If you have worked through this example, you can see that all the buttons in the Edit column (Edit, Update, Cancel) are shown as links. This is the default setting, but you can change it so that these items appear either as buttons or custom images. To show these items as buttons, you simply set the `ButtonType` attribute to `Button`:

```
<asp:CommandField ShowEditButton="True" ButtonType="Button">
</asp:CommandField>
```

The results of this are illustrated in Figure 4-12.

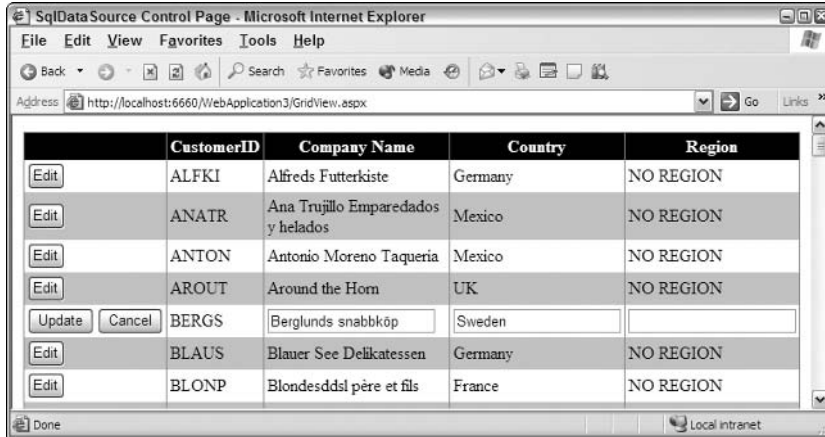


Figure 4-12

You can also use custom images for these actions. To do this, you specify `Image` as the `ButtonType`:

```
<asp:CommandField ShowEditButton="True" ButtonType="Image"
  EditImageUrl="~/edit.gif" UpdateImageUrl="~/fix.gif"
  CancelImageUrl="~/cancel.gif">
</asp:CommandField>
```

If you instruct the `ButtonType` to use images, you also use the `EditImageUrl`, `UpdateImageUrl`, and the `CancelImageUrl` attributes to give the location of the images you want in the table. Doing this produces the following results, as illustrated in Figure 4-13.

In the example in Listing 4-9, another change made to the columns and their appearance in the table involves the `CustomerID` column. The data elements contained within this column uniquely identify the customer and, in this case, it is actually the primary key for the entry in the database. Because of this, you don't want the end user to edit this entry. Turning off the edit capability is easy to do by using the `ReadOnly` attribute for the column definition:

```
<asp:BoundField SortExpression="CustomerID" HeaderText="CustomerID"
  ReadOnly="True" DataField="CustomerID">
</asp:BoundField>
```

	CustomerID	Company Name	Country	Region
Edit	ALFKI	Alfreds Futterkiste	Germany	NO REGION
Edit	ANATR	Ana Trujillo Emparedados y helados	Mexico	NO REGION
Edit	ANTON	Antonio Moreno Taquería	Mexico	NO REGION
Edit	AROUT	Around the Horn	UK	NO REGION
Fix Back	BERGS	Berglunds snabbköp	Sweden	
Edit	BLAUS	Blauer See Delikatessen	Germany	NO REGION
Edit	BLONP	Blondesddsl père et fils	France	NO REGION
Edit	BOLID	Bólido Comidas preparadas	Spain	NO REGION
Edit	BONAP	Bon app'	France	NO REGION

Figure 4-13

Specifying the `ReadOnly` attribute as `True` means that although the end user can edit some rows in the table, he can't edit the `CustomerID` field. This is shown in the previous screen shots of this table.

Now that the columns specified in the `<Columns>` section of the `GridView` control are in place, you must associate the primary key to identify a row when it is sent back to the server for an update or deletion. This is done in the main `<asp:GridView>` element:

```
<asp:gridview id="GridView1" Runat="server" DataSourceId="SqlDataSource1"
AutoGenerateColumns="False" DataKeyNames="CustomerID"
BackColor="White" GridLines="Vertical"
BorderStyle="Solid" CellPadding="3" ForeColor="Black"
BorderColor="#999999" BorderWidth="1px">
```

You make this association with the `DataKeyNames` attribute (shown in bold). You can see that the value assigned to this attribute in the example points to the `CustomerID` field. Without this specification, the selected row is not updated. After this is in place, the `GridView` control is ready. Now turn your attention to the `SqlDataSource` control used by the `GridView` control.

You want the table not only to display data (using the `SelectCommand` attribute), but also to enable the end user to push updates of the data to SQL Server. You do this by adding an `UpdateCommand` attribute to the `SqlDataSource` control:

```
<asp:SqlDataSource Id="SqlDataSource1" Runat="server"
SelectCommand="Select * From Customers"
UpdateCommand="UPDATE Customers SET CompanyName = @CompanyName,
Country = @Country, Region = @Region WHERE (CustomerID = @CustomerID)"
ConnectionString="Server=(local);Trusted_Connection=True;Integrated
Security=SSPI;Persist Security Info=True;Database=Northwind"
ProviderName="System.Data.SqlClient">
```

The `UpdateCommand` attribute takes a `String` value, which is the SQL command to update the database. A number of named parameters, such as `@CompanyName`, `@Country`, `@Region`, and `@CustomerID`, are placeholders for the corresponding information that is going to come from the selected row. Now that you are using named parameters in the update command, however, you must define the parameters. This is where the `<UpdateParameters>` section of the `SqlDataSource` control comes into play:

```
<UpdateParameters>
  <asp:Parameter Name="CustomerID" Type="String">
  </asp:Parameter>
  <asp:Parameter Name="CompanyName" Type="String">
  </asp:parameter>
  <asp:parameter Name="Country" Type="String">
  </asp:parameter>
  <asp:parameter Name="Region" Type="String">
  </asp:parameter>
</UpdateParameters>
```

Within the `<UpdateParameters>` section of the `GridView` control, each named parameter is defined using the `<asp:Parameter>` element. The `<asp:Parameter>` element here uses two attributes that define the name and data type of the parameter (in this case, all parameters are of type `String`). In addition to an `<UpdateParameters>` section for the `GridView` control, you can also use the `<DeleteParameters>`, `<FilterParameters>`, `<InsertParameters>`, and `<SelectParameters>` sections to provide parameters for other operations.

Deleting data from the GridView

Deleting data from the table produced by the `GridView` is even easier than editing data. Just a few additions to the code enable you to delete an entire row of data from the table. As you saw in the previous example (Listing 4-9), you simply add the `Delete` button to the `<asp:CommandField>` element of the `GridView` by setting the value of the `ShowDeleteButton` attribute to `True` (see Listing 4-10).

Listing 4-10: Adding a Delete link (partial code)

```
<asp:CommandField ShowEditButton="True" ShowDeleteButton="True">
</asp:CommandField>
```

The addition of the `ShowDeleteButton` to the `GridView` is the only change you make to this control. Now look at the `SqlDataSource` control. Listing 4-11 shows you the root element of this control.

Listing 4-11: Adding delete functionality to the SqlDataSource control

```
<asp:SqlDataSource Id="SqlDataSource1" Runat="server"
  SelectCommand="Select * From Customers"
  UpdateCommand="UPDATE Customers SET CompanyName = @CompanyName,
    Country = @Country, Region = @Region WHERE (CustomerID = @CustomerID)"
  DeleteCommand="DELETE From Customers WHERE (CustomerID = @CustomerID)"
  ConnectionString="Server=(local);Trusted_Connection=True;Integrated
    Security=SSPI;Persist Security Info=True;Database=Northwind"
  ProviderName="System.Data.SqlClient">
```

In addition to the `SelectCommand` and `UpdateCommand` attributes, you also add the `DeleteCommand` attribute to the `SqlDataSource` server control and provide the SQL command that deletes the specified row. Just like the `UpdateCommand` attribute, the `DeleteCommand` attribute uses a named parameter. Because of this, you define what this parameter is from within the `SqlDataSource` control. To do this, add a `<DeleteParameters>` section to the `SqlDataSource` control, as shown in Listing 4-12.

Listing 4-12: Adding a `<DeleteParameters>` section to the `SqlDataSource` control

```
<DeleteParameters>
  <asp:Parameters Name="CustomerID" Type="String">
  </asp:Parameters>
</DeleteParameters>
```

This is the only parameter definition needed for the `<DeleteParameters>` section because the SQL command for this deletion requires only the `CustomerID` from the row to delete the entire row.

When you run the example with this code in place, you see a Delete link next to the Edit link. Clicking the Delete link completely deletes the selected row.

Dealing with other column types in the GridView

So far in this chapter, you have looked at only two types of columns in the `GridView` control: the `BoundField` and the `CommandField` columns. Although these are the column types that you most commonly use, additional column types are at your disposal (such as the `CheckBoxField`, `HyperLinkField`, and the `TemplateField` columns). Take a quick look at each of these column types.

`<asp:CheckBoxField>`

An example of the `CheckBoxField` column is the `Discontinued` column, shown in Figure 4-14. In it, the `GridView` control displays the contents of the `Products` table in the Northwind database of SQL Server.

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0	10	<input type="checkbox"/>
2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40	25	<input type="checkbox"/>
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70	25	<input type="checkbox"/>
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.0000	53	0	0	<input type="checkbox"/>
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.3500	0	0	0	<input checked="" type="checkbox"/>
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.0000	120	0	25	<input type="checkbox"/>
	Tia's Bob's								

Figure 4-14

The GridView automatically converts a column type to a `CheckBoxField` if the data store's column type is using a `Boolean` or a bit value. To specify which columns should be displayed using the `<Columns>` section of the GridView, use the method shown in Listing 4-13.

Listing 4-13: Showing check boxes in the GridView control

```
<Columns>
  <asp:CheckBoxField SortExpression="Discontinued" DataField="Discontinued"
    HeaderText="Discontinued">
  </asp:CheckBoxField>
</Columns>
```

<asp:HyperLinkField>

You can also just as easily create columns in your tables that include hyperlinks by using the `HyperLinkField` column. You can use the `HyperLinkField` column to enable end users to see related information on another page that you can link to from each row in the table. Each row might need a distinct URL, requiring you to put this type of column in the GridView. Listing 4-14 shows an example of using the `<asp:HyperLinkField>` element to link to more information about the customer represented in the Customers table from the Northwind database of SQL Server.

Listing 4-14: Showing hyperlinks in the GridView control

```
<Columns>
  <asp:HyperLinkField SortExpression="CompanyName" DataTextField="CompanyName"
    HeaderText="Company Name" DataNavigateUrlFields="CustomerID"
    DataNavigateUrlFormatString="http://www.reuters.com/mycustomers?custid={0}">
  </asp:HyperLinkField>
</Columns>
```

This code gives you the results illustrated in Figure 4-15.

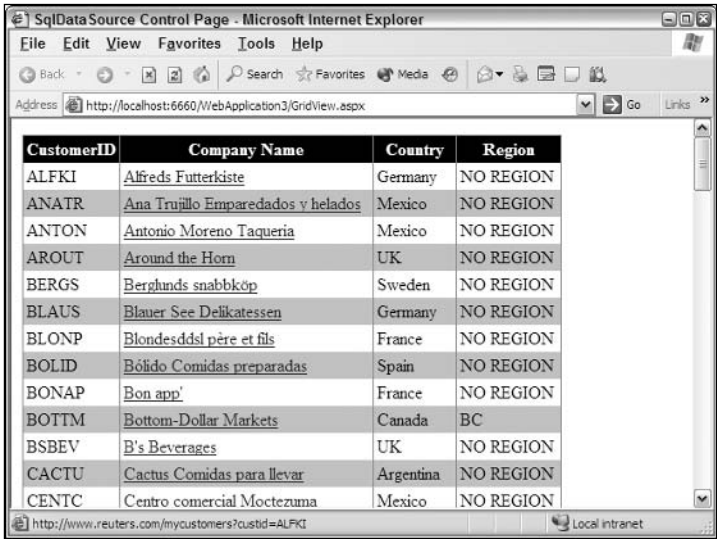


Figure 4-15

You can see that each of the company names is a hyperlink in the table. If you hover your mouse over the links, note that each one has a different hyperlink destination. The `<asp:HyperLinkField>` element contains some attributes that differ from the other items in the `<Columns>` section of the `GridView` control. Instead of a `DataField` attribute, the `HyperLinkField` column uses a `DataTextField` attribute. This attribute specifies the text used for the hyperlink — in this case, the information found in the `CompanyName` column from the data store.

Using the `DataNavigateUrlFormatString` attribute, you specify the link destination to be used for the hyperlink. In this case, it is `http://www.reuters.com/mycustomers?custid={0}`. Looking at the link itself, you see that every customer link goes to the same page, but the page can be customized for the customer selected in the table because of the `QueryString` used with the link (`custid={0}`). The `QueryString` itself does not contain a value for `custid`. This is generated dynamically and is unique for each and every row in the table. You show the dynamic part of a hyperlink with the use of `{0}`.

After the `DataNavigateUrlFormatString` attribute is in place, you specify what takes the place of the `{0}` in each hyperlink that is created. For this, you use the `DataNavigateUrlFields` attribute. This attribute can take one or more fields to be used in the hyperlink. For this example, refer to the `CustomerID` column from the data store. When you refer to `CustomerID`, a link like this appears: `http://www.reuters.com/mycustomers?custid=WARTH`.

By the way, if you have to create a hyperlink such as `http://www.reuters.com/mycustomers?custid=WARTH&country=Finland`, you give the `DataNavigateUrlFormatString` a value of `http://www.reuters.com/mycustomers?custid={0}&country={1}`. Because this hyperlink contains two dynamic parts, you construct the `DataNavigateUrlFields` attribute as follows:

```
DataNavigateUrlFields="CustomerID, Country"
```

<asp:TemplateField>

The final column type is the `TemplateField` column. This column enables you to completely customize the appearance and the structure of the generated column in the `GridView` control. You can place pretty much anything you want in the cells of your table. For instance, you could put any of the other `<asp:>` controls in the templates used for the cells — this would be easy to achieve using the `<asp:TemplateField>`. For example, if you want a cell that contains both the `CustomerID` and `CompanyName` items from the `Customers` table in the `Northwind` database, you use the code illustrated in Listing 4-15.

Listing 4-15: Working with the `TemplateField` column

```
<asp:TemplateField SortExpression="CustomerID" HeaderText="Our Customers">
  <itemtemplate>
    <asp:Label Runat="server" Text='<%# Eval("CustomerID") %>'
      Font-Bold="True" BackColor="Red" /><br />
    <asp:Label Runat="Server" Text='<%# Eval("CompanyName") %>' />
  </itemtemplate>
  <alternatingitemtemplate>
    <asp:Label Runat="server" Text='<%# Eval("CustomerID") %>'
      Font-Bold="True" BackColor="Yellow" /><br />
    <asp:Label ID="Label1" Runat="Server" Text='<%# Eval("CompanyName") %>' />
  </alternatingitemtemplate>
</asp:TemplateField>
```


Chapter 4

In this example, both the `CustomerID` data item and the `CompanyName` data item are shown in the same cell. By using the `<ItemTemplate>` and the `<AlternatingItemTemplate>` elements, you can change the background color used for the `CustomerID` data item, as shown in Figure 4-16 (it's not easy to tell in this black-and-white screen shot, but red and yellow have been used as background colors for the customer ID).

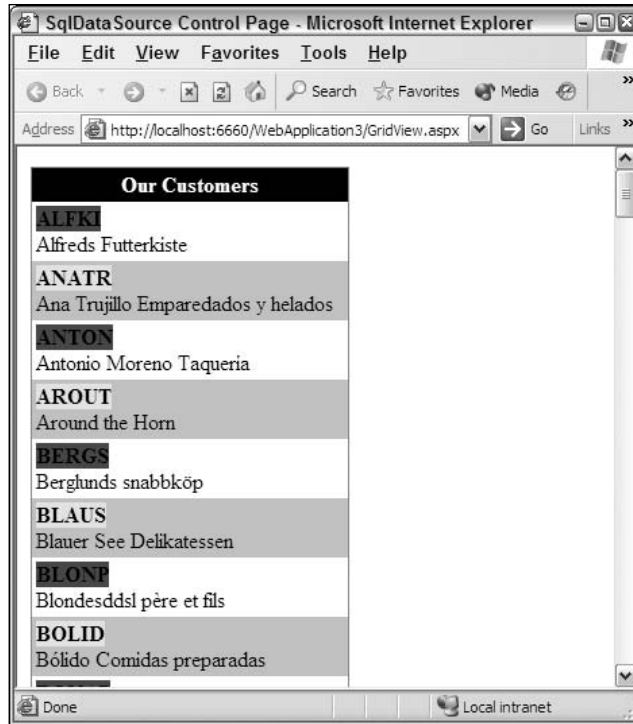


Figure 4-16

The AccessDataSource and DetailsView Controls

Not only can you easily connect to Microsoft SQL Server using the new `SqlDataSource` control, you can also connect to any Microsoft Access database (an `.mdb` file) just as easily by using the `AccessDataSource` control provided with ASP.NET 2.0. Microsoft Access is a great database to use in smaller sites or applications. It is especially useful when you are working with an application hosted on a third-party server that doesn't provide or allow you to work with a SQL Server database. You can use this data source control with any of the data-bound server controls. Previous examples of using the `GridView` control can be done just as easily using the `AccessDataSource` control in place of the `SqlDataSource` control. Instead of showing more examples using just the `GridView` control, this section focuses on using the `AccessDataSource` control with another new data-bound control — the `DetailsView` control.

You won't find that much difference between the `AccessDataSource` control and the previously viewed `SqlDataSource` control (besides the fact that they connect to different types of data stores). You can work with the `AccessDataSource` control in much the same manner as you worked with the `SqlDataSource` control.

Microsoft Access, just like SQL Server, comes with a sample database called Northwind. For this chapter, I use the sample Northwind database found at `C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb` from the Microsoft Access product that is part of Office 11. You can get to the sample Northwind database within Microsoft Access by choosing `Help ⇨ Sample Databases ⇨ Northwind Sample Database`. If you do not have the database installed, you are prompted at this time to install it.

After the sample database is in place, you can use this sample `.mdb` file (titled `Northwind.mdb`) and the `AccessDataSource` control to work with the samples in this chapter.

The `DetailsView` control is a new data-bound control that enables you to view a single record at a time. Although the `GridView` control is an excellent control for viewing a collection of data that consists of multiple rows, you might also want to drill down into one of those individual records. The `DetailsView` control lets you do this.

This control has a lot of the same types of behaviors and functionality found in the `GridView` control. You can use the `DetailsView` control to do things like paging, updating, inserting, and deleting data.

For an example of using the new `AccessDataSource` control with the new `DetailsView` control, follow these steps to get at the Customers table from the Northwind database in the sample `.mdb` file:

1. Create a new ASP.NET page using either Visual Basic or C#.
2. Drag and drop a `DetailsView` server control onto the design surface.
3. Change the style of the `DetailsView` to something more interesting by clicking the `AutoFormat` link in control's smart tag (for this example, I selected `Brown Sugar`).
4. Add the `Northwind.mdb` file to your solution by right-clicking the solution and selecting `Add Existing Item`. You can find the sample `.mdb` file at `C:\Program Files\Microsoft Office\OFFICE11\SAMPLES\Northwind.mdb`. It is best to place this and all `.mdb` files within the `Data` folder included with your solution.
5. Drag and drop an `AccessDataSource` control onto the design surface.

Just like the `SqlDataSource` control, the `AccessDataSource` control is represented on the Design surface as a gray box. Highlight the gray box and turn your attention to the Properties Window.

The first item to change here is the `DataFile` property. This property should point to the `Northwind.mdb` file. You can do this by simply giving the `DataFile` property a value of `Northwind.mdb`.

Next, change the `SelectQuery` property by clicking the button next to this property. This launches the Command and Parameter Editor dialog, as shown in Figure 4-17.

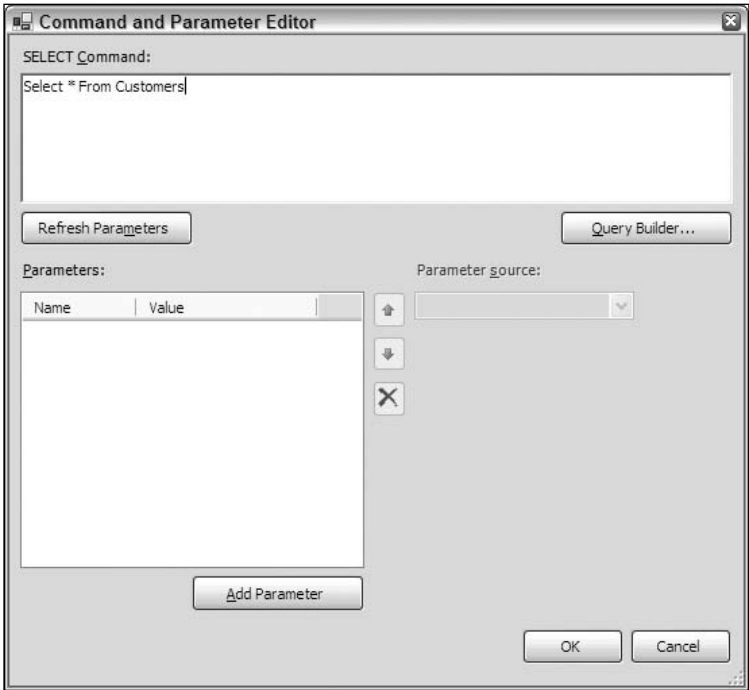


Figure 4-17

Place the value `Select * From Customers` in the `SELECT Command` text box. Click `OK`. The `AccessDataSource` control is now ready to use.

Next, locate `DetailsView` on the `Design` surface, highlight the `DetailsView1` control on the page, and give the `DataSourceId` property the value of `AccessDataSource1`. If you run the page at this point, the `DetailsView` control displays only a single result (the first result) and won't allow you to see anything else. If that is the functionality you are looking for, you probably want a different `SQL` statement, such as `Select * From Customers Where CustomerID=ALFKI`, so that you don't grab every record from the table.

To enable the end user to scroll through the records one at a time, you enable paging in the `DetailsView` control. This is done either by adding `AllowPaging="True"` or by checking the `Enable Paging` check box in the smart tag of the `DetailsView` control (shown in Figure 4-18).

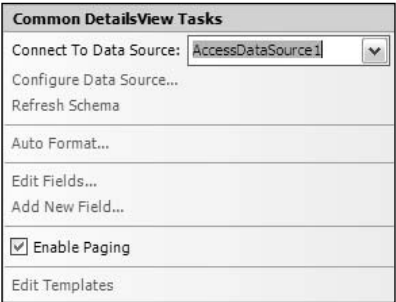


Figure 4-18

After all this is complete, your code should resemble Listing 4-16.

Listing 4-16: Looking at a single record at a time

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>DetailsView Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:DetailsView ID="DetailsView1" Runat="server"
            DataSourceID="AccessDataSource1"
            AllowPaging="True" BorderColor="#DEBA84" BorderStyle="None"
            BorderWidth="1px"
            BackColor="#DEBA84" CellSpacing="2" CellPadding="3">
            <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center">
            </PagerStyle>
            <EditRowStyle ForeColor="White" BackColor="#738A9C" Font-Bold="True">
            </EditRowStyle>
            <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7">
            </RowStyle>
            <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
            </FooterStyle>
            <HeaderStyle ForeColor="White" BackColor="#A55129" Font-Bold="True">
            </HeaderStyle>
        </asp:DetailsView>
        <asp:AccessDataSource ID="AccessDataSource1" Runat="server"
            SelectCommand="Select * From Customers"
            DataFile="Data/Northwind.mdb">
        </asp:AccessDataSource>
    </form>
</body>
</html>
```

After it is run, your page should look like Figure 4-19.

The DetailsView control associates itself with the AccessDataSource control via the use of the DataSourceID attribute. This attribute takes a String value and points to a particular data source control by using the data source control's control ID as a value. In this case, it is AccessDataSource1.

The AccessDataSource control, shown previously, uses the typical ID and Runat attributes as well as the SelectCommand attribute. The SelectCommand attribute takes a String value. Later when you begin inserting, updating, and deleting data from a data store using the DetailsView control, you use the other InsertCommand, UpdateCommand, and DeleteCommand attributes. The last important attribute from Listing 4-16 is the DataFile attribute. This attribute takes a String value that points to the location of the Access data file used by the control. In this case, because the Northwind.mdb file is located with the .aspx file itself, the value of this attribute is simply Northwind.mdb.

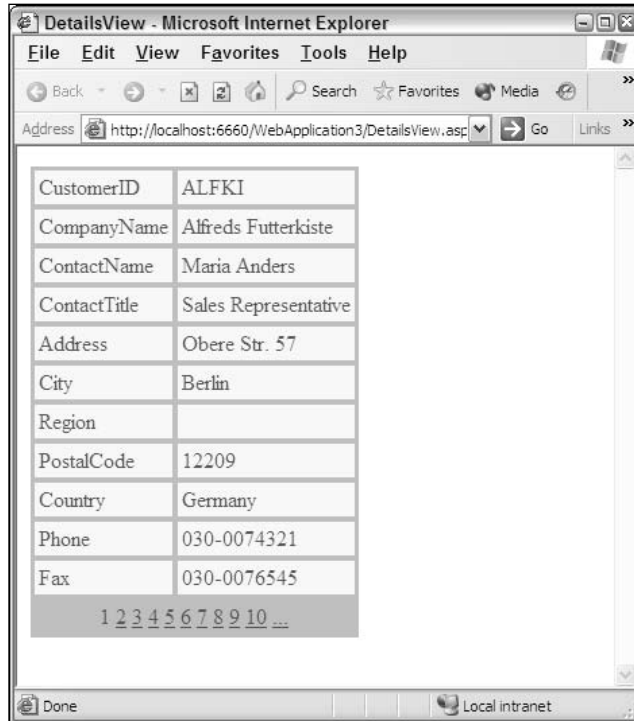


Figure 4-19

Looking at the figure, you can see that, by default, the DetailsView control performs its paging capability in a numeric fashion. You can, however, change this behavior just as you can change the behavior of the GridView control. Just add the following attribute to the root `<asp:DetailsView>` element in the DetailsView control:

```
PagerSettings-Mode="NextPrevious"
```

This gives you logical arrows that enable the end user to move up or down the row collection, as illustrated in Figure 4-20.

Also, just as with the GridView control, you can completely customize not only the appearance of the pager functionality (including the use of images), but also the appearance of the entire control.

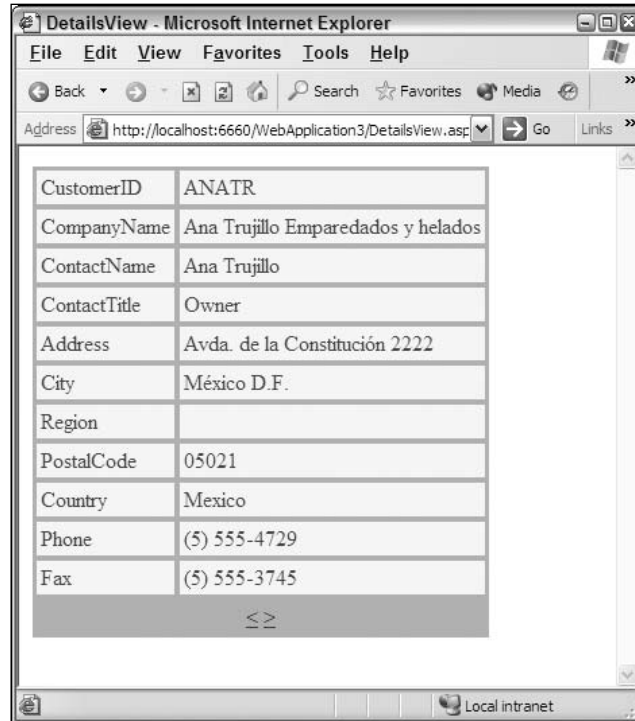


Figure 4-20

Selecting which fields to display in the DetailsView control

You can customize the appearance of the DetailsView control by picking and choosing which fields the control displays. By default, the control displays each and every column from the table it is working with. Much like the GridView control, however, the DetailsView control enables you to specify that only certain selected columns be displayed. This is illustrated in Listing 4-17.

Listing 4-17: Choosing the fields to display in the DetailsView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>DetailsView Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:DetailsView ID="DetailsView1" Runat="server">
```

(continued)

Listing 4-17: (continued)

```
DataSourceId="AccessDataSource1"
AllowPaging="True" BorderColor="#DEBA84" BorderStyle="None"
BorderWidth="1px" BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
PagerSettings-Mode="NextPrevious" AutoGenerateRows="False">
<Fields>
  <asp:BoundField HeaderText="Customer ID" DataField="CustomerID">
  </asp:BoundField>
  <asp:BoundField HeaderText="Company Name" DataField="CompanyName">
  </asp:BoundField>
  <asp:BoundField HeaderText="Country" DataField="Country">
  </asp:BoundField>
  <asp:BoundField NullDisplayText="NO REGION" HeaderText="Region"
    DataField="Region">
  </asp:BoundField>
</Fields>
<PagerStyle ForeColor="#8C4510" HorizontalAlign="Center">
</PagerStyle>
<EditRowStyle ForeColor="White" BackColor="#738A9C" Font-Bold="True">
</EditRowStyle>
<RowStyle ForeColor="#8C4510" BackColor="#FFF7E7">
</RowStyle>
<FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
</FooterStyle>
<HeaderStyle ForeColor="White" BackColor="#A55129" Font-Bold="True">
</HeaderStyle>
</asp:DetailsView>
<asp:AccessDataSource ID="AccessDataSource1" Runat="server"
  SelectCommand="Select * From Customers"
  DataFile="Data/Northwind.mdb">
</asp:AccessDataSource>
</form>
</body>
</html>
```

When this page is run, you see that the DetailsView control now displays only four values for each of the rows (as shown in Figure 4-21).

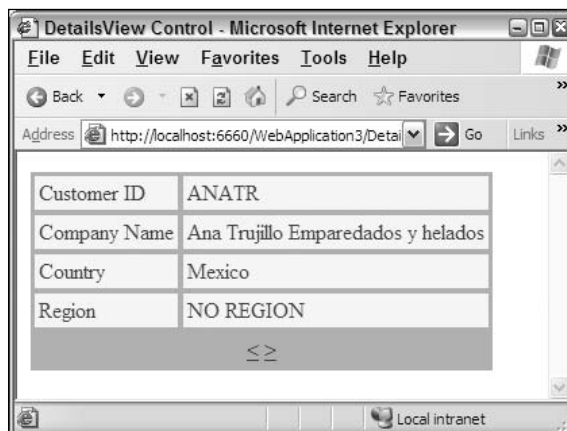


Figure 4-21

You select columns exactly as you did when using the GridView control. The big difference is that instead of using the <Columns> element to define the columns to bind to, the DetailsView control uses the <Fields> element. Notice that the columns are bound to the DetailsView control using the same ASP.NET controls you used for the GridView control: <asp:BoundField>, <asp:ButtonField>, <asp:CheckBoxField>, <asp:CommandField>, <asp:HyperLinkField>, and <asp:TemplateField>. In fact, you use these field declaration controls just as you did with the GridView control shown earlier in the chapter.

Using the GridView and DetailsView together

Now for an interesting and useful example, look at how you would use the GridView and the DetailsView controls together. You can use the GridView to show a master view of the data (in this case, the Customers table from the sample Northwind.mdb file), whereas you can use the DetailsView control to show the details of any selected rows in the main GridView control (see Listing 4-18).

Listing 4-18: Enabling the GridView and DetailsView controls to work together

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>GridView & DetailsView Controls</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:GridView ID="GridView1" Runat="server"
                DataSourceId="AccessDataSource1" AllowPaging="True"
                BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px"
                BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
                DataKeyNames="CustomerID" AutoGenerateSelectButton="true"
                AutoGenerateColumns="False" PageSize="5">
                <Columns>
                    <asp:BoundField HeaderText="Customer ID"
                        DataField="CustomerID">
                    </asp:BoundField>
                    <asp:BoundField HeaderText="Company Name"
                        DataField="CompanyName">
                    </asp:BoundField>
                </Columns>
                <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
                </FooterStyle>
                <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7">
                </RowStyle>
                <SelectedRowStyle ForeColor="White" BackColor="#738A9C"
                    Font-Bold="True">
                </SelectedRowStyle>
                <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center">
                </PagerStyle>
                <HeaderStyle ForeColor="White" BackColor="#A55129"
                    Font-Bold="True">
                </HeaderStyle>
            </asp:GridView>
        </p>
    </form>
</body>
</html>
```

(continued)

Listing 4-18: (continued)

```
</asp:GridView>
</p>
<p><b>Customer Details:</b></p>
<asp:DetailsView ID="DetailsView1" Runat="server"
  DataSourceID="AccessDataSource2"
  BorderColor="#DEBA84" BorderStyle="None" BorderWidth="1px"
  BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
  AutoGenerateRows="True">
  <PagerStyle ForeColor="#8C4510" HorizontalAlign="Center">
  </PagerStyle>
  <EditRowStyle ForeColor="White" BackColor="#738A9C" Font-Bold="True">
  </EditRowStyle>
  <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7">
  </RowStyle>
  <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
  </FooterStyle>
  <HeaderStyle ForeColor="White" BackColor="#A55129" Font-Bold="True">
  </HeaderStyle>
</asp:DetailsView>
<asp:AccessDataSource ID="AccessDataSource1" Runat="server"
  SelectCommand="Select * From Customers"
  DataFile="~/Northwind.mdb" />
<asp:AccessDataSource ID="AccessDataSource2" Runat="server"
  SelectCommand="Select * From Customers"
  DataFile="~/Northwind.mdb" FilterExpression="CustomerID='@CustID'">
  <FilterParameters>
    <asp:ControlParameter Name="CustID" ControlId="GridView1"
      PropertyName="SelectedValue">
    </asp:ControlParameter>
  </FilterParameters>
</asp:AccessDataSource>
</form>
</body>
</html>
```

When this page is run in the browser, you get the results illustrated in Figure 4-22.

In this figure, one of the rows in the GridView has been selected (noticeable by the color change). The details of this selected row are shown in the DetailsView control directly below the GridView control.

To see how this works, look at the changes made to the second AccessDataSource control, AccessDataSource2. The only change made to the main <asp:AccessDataSource> element here is the addition of the FilterExpression attribute. The FilterExpression attribute is used to modify the SelectCommand attribute when filtering is applied. The String value given to the FilterExpression attribute expresses how you want the AccessDataSource control to filter the select command. In this case, the value of the FilterExpression is CustomerID=@CustomerID. If filtering is utilized, the AccessDataSource control filters the records that are retrieved from the Select * From Customers command so that the command resembles Select * From Customers Where CustomerID=@CustomerID. The parameter specified in the FilterExpression attribute, @CustomerID, is defined within the AccessDataSource control itself through the use of the <FilterParameters> element.

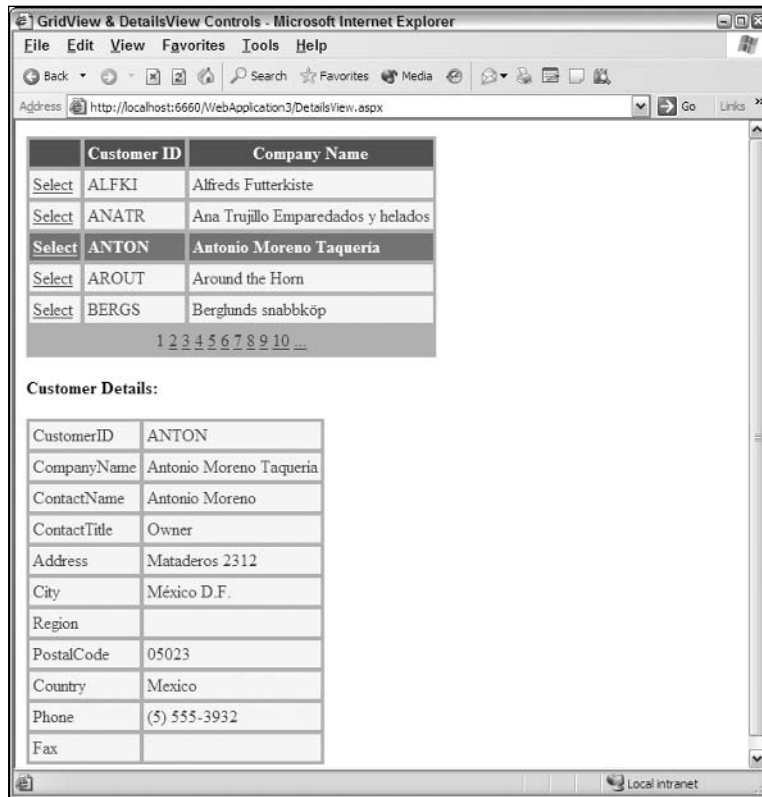


Figure 4-22

Using the `<FilterParameters>` element within the `AccessDataSource` control is the same as using it with the `SqlDataSource` control. In this case, the `AccessDataSource` control in Listing 4-18 uses an `<asp:ControlParameter>` to specify the name of the parameter, the control that the parameter value is coming from (the `GridView1` control), and the property name from the control that is used to populate the parameters value.

In the `GridView1` control, be sure to include the `DataKeyNames` attribute. This control gives this attribute a value of `CustomerID` — meaning that when the end user selects one of the rows in the table, the `CustomerID` value for that chosen row becomes what is provided via the `SelectedValue` property.

Updating, inserting, and deleting rows

Updating rows in the `DetailsView` control is quite similar to updating rows in the `GridView` — but with a few little twists that you should be aware of. The process for inserting and deleting rows is very similar to that for updating rows (shown here). First, to update data being displayed by the `DetailsView` control, you modify the `<asp:AccessDataSource>` control so that it allows for the editing of the data to which it is connected. The main node of the `<asp:AccessDataSource>` element is shown in Listing 4-19.

Listing 4-19: The <asp:AccessDataSource> element

```
<asp:AccessDataSource ID="AccessDataSource1" Runat="server"
  SelectCommand="Select * From Customers"
  UpdateCommand="UPDATE [Customers] SET [CompanyName] = ?, [ContactName] = ?,
    [ContactTitle] = ?, [Address] = ?, [City] = ?, [Region] = ?, [PostalCode] = ?,
    [Country] = ?, [Phone] = ?, [Fax] = ? WHERE [CustomerID] = ?"
  DataFile="Data/Northwind.mdb" FilterExpression="CustomerID='@CustID'">
  ...
</asp:AccessDataSource>
```

A few things have been added to the AccessDataSource control to enable it to edit the data from the Northwind.mdb file. The first addition is that the UpdateCommand attribute has been added with a SQL command that updates the data store based upon a large collection of parameters that I define shortly.

The next required change to the AccessDataSource control is the addition of the update parameter definitions. You do this by using the <UpdateParameters> element within the AccessDataSource control itself. This is illustrated in Listing 4-20.

Listing 4-20: Adding the Update parameters to the AccessDataSource control

```
<UpdateParameters>
  <asp:Parameter Type="String" Name="CompanyName"></asp:Parameter>
  <asp:Parameter Type="String" Name="ContactName"></asp:Parameter>
  <asp:Parameter Type="String" Name="ContactTitle"></asp:Parameter>
  <asp:Parameter Type="String" Name="Address"></asp:Parameter>
  <asp:Parameter Type="String" Name="City"></asp:Parameter>
  <asp:Parameter Type="String" Name="Region"></asp:Parameter>
  <asp:Parameter Type="String" Name="PostalCode"></asp:Parameter>
  <asp:Parameter Type="String" Name="Country"></asp:Parameter>
  <asp:Parameter Type="String" Name="Phone"></asp:Parameter>
  <asp:Parameter Type="String" Name="Fax"></asp:Parameter>
  <asp:Parameter Type="String" Name="CustomerID"></asp:Parameter>
</UpdateParameters>
```

Each of these parameters defines what is used in the UpdateCommand string. Now that the AccessDataSource control is ready, turn your attention to the DetailsView control.

You need to make only a few changes to the DetailsView control. First, you add the Edit button to the control just as you added it to the GridView control. This is illustrated in Listing 4-21.

Listing 4-21: Adding an Edit button to the DetailsView control

```
<asp:DetailsView ID="DetailsView1" Runat="server"
  DataSourceID="AccessDataSource1" DataKeyNames="CustomerID"
  AllowPaging="True" BorderColor="#DEBA84" BorderStyle="None"
  BorderWidth="1px" PagerSettings-Mode="NextPrevious"
  BackColor="#DEBA84" CellSpacing="2" CellPadding="3"
  AutoGenerateRows="True" AutoGenerateEditButton="True">
  ...
</asp:DetailsView>
```

You add an Edit button to the DetailsView control via the use of the `AutoGenerateEditButton` attribute, which you set to `True`. You should also add a `DataKeyNames` attribute, which points to the `IDENTITY` field that should be utilized for the update. In this case, it is the `CustomerID` field. After this is in place and running, the DetailsView control appears with an Edit link at the bottom of the control (as shown in Figure 4-23).

Customer Details:	
CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545
Edit	
≡	

Figure 4-23

If the end user clicks on the Edit hyperlink, most of the fields within the DetailsView control are changed to allow for editing, and the Edit hyperlink is replaced with an Update and Cancel hyperlink, as shown in Figure 4-24.

Adding the capability to insert or delete rows using the `AccessDataSource` control is the same as using the `SqlDataSource` control as shown earlier in the chapter.

The DetailsView control also works like GridView control in regard to inserting and deleting data. To insert additional rows into the Access file using the DetailsView control, simply add the `AutoGenerateInsert` attribute to the control:

```
AutoGenerateInsertButton="True"
```

After this is in place, you find a New hyperlink next to the Edit hyperlink, as illustrated in Figure 4-25.

Customer Details:	
CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545
Update Cancel	
≥	

Figure 4-24

Customer Details:	
CustomerID	ALFKI
CompanyName	Alfreds Futterkiste
ContactName	Maria Anders
ContactTitle	Sales Representative
Address	Obere Str. 57
City	Berlin
Region	
PostalCode	12209
Country	Germany
Phone	030-0074321
Fax	030-0076545
Edit New	
≥	

Figure 4-25

Clicking the New hyperlink provides a table that enables you to enter a new customer after you add an `INSERT` command to the `AccessDataSource` control just as was done with the `UPDATE` command. This is shown in Figure 4-26.

Customer Details:	
CustomerID	<input type="text"/>
CompanyName	<input type="text"/>
ContactName	<input type="text"/>
ContactTitle	<input type="text"/>
Address	<input type="text"/>
City	<input type="text"/>
Region	<input type="text"/>
PostalCode	<input type="text"/>
Country	<input type="text"/>
Phone	<input type="text"/>
Fax	<input type="text"/>
Insert Cancel	

Figure 4-26

Be sure that if you enable the inserting of data into your data store, that you define the `<InsertParameters>` using the `<asp:Parameter>` control — just as was done in the updating of the data.

XmlDataSource Control

So far, you have been looking at using the data source controls that work with traditional data stores such as Microsoft SQL Server and Microsoft Access. Today, however, a considerable amount of data is stored in XML format, so a specific data source control has been added to ASP.NET 2.0 just for retrieving and working with XML data.

The `XmlDataSource` control enables you to connect to your XML data and to use this data with any of the ASP.NET data-bound controls. Just like the `SqlDataSource` and the `AccessDataSource` controls, the `XmlDataSource` control also enables you to not only retrieve data, but also to insert, delete, and update data items. With the world turning more and more to XML data formats, such as Web services, RSS feeds, and more, this control is a valuable resource for your applications.

To show the `XmlDataSource` control in action, first create a simple XML file and include this file in your application. Listing 4-22 shows a simple XML file of Russian painters that we can use.

Listing 4-22: Painters.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Artists>
  <Painter name="Vasily Kandinsky">
    <Painting>
      <Title>Composition No. 218</Title>
      <Year>1919</Year>
    </Painting>
  </Painter>
  <Painter name="Pavel Filonov">
    <Painting>
      <Title>Formula of Spring</Title>
      <Year>1929</Year>
    </Painting>
  </Painter>
  <Painter name="Pyotr Konchalovsky">
    <Painting>
      <Title>Sorrento Garden</Title>
      <Year>1924</Year>
    </Painting>
  </Painter>
</Artists>
```

Now that the `Painters.xml` file is in place, the next step is to use a `DataList` control and connect this `DataList` control to an `<asp:XmlDataSource>` control. This is illustrated in Listing 4-23.

Listing 4-23: Using a `DataList` control to display XML content

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>XmlDataSource</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:DataList ID="DataList1" Runat="server" DataSourceID="XmlDataSource1">
      <ItemTemplate>
        <p><b><%# XPath("@name") %></b><br />
          <i><%# XPath("Painting/Title") %></i><br />
          <%# XPath("Painting/Year") %></p>
      </ItemTemplate>
    </asp:DataList>

    <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
      DataFile="~/Painters.xml" XPath="Artists/Painter">
    </asp:XmlDataSource>
  </form>
</body>
</html>
```

This is a simple example, but it shows you the power and ease of using the `XmlDataSource` control. You should pay attention to only two attributes in this example. The first is the `DataFile` attribute. This attribute points to the location of the XML file. Because the file resides in the root directory of the application, it is simply `~/Painters.xml`. The next attribute included in the `XmlDataSource` control is the `XPath` attribute. The `XmlDataSource` control uses XPath for the filtering of XML data. In this case, the `XmlDataSource` control is taking everything within the `<Painter>` set of elements. The value `Artists/Painter` means that the `XmlDataSource` control navigates to the `<Artists>` element and then to the `<Painter>` element within the specified XML file.

The `DataList` control next must specify the `DataSourceID` as the `XmlDataSource` control. In the `<ItemTemplate>` section of the `DataList` control, you can retrieve specific values from the XML file by using XPath commands. The XPath commands filter the data from the XML file. The first value retrieved is an element attribute (`name`) that is contained in the `<Painter>` element. If you are retrieving an attribute of an element, you preface the name of the attribute with an `@` symbol. In this case then, you simply specify `@name` to get at the painter's name. The next two XPath commands go deeper into the XML file and get the specific painting and the year of the painting. Remember to separate nodes with a `/`. When run in the browser, this code produces the results illustrated in Figure 4-27.

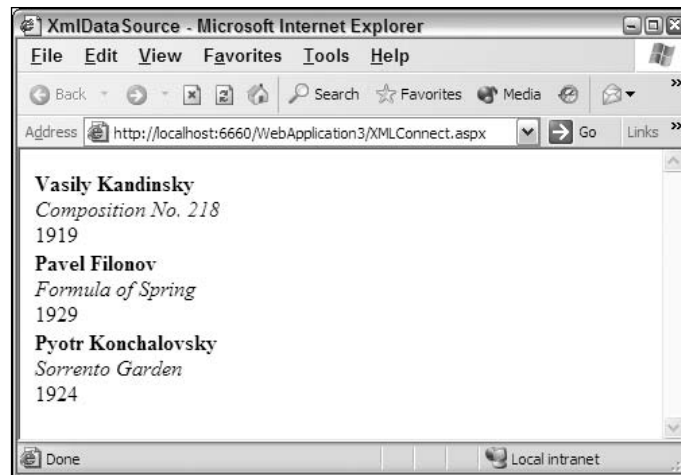


Figure 4-27

Besides working from static XML files like the `Painters.xml` file shown earlier, the `XmlDataSource` file has the capability to work from dynamic, URL-accessible XML files. One popular XML format that is pervasive on the Internet today is blogs or weblogs. Blogs, or personal diaries, can be viewed either in the browser, through an RSS-aggregator, or just as pure XML.

As you look at my blog in Figure 4-28, you can see the XML it produces directly in the browser. (You can find a lot of blogs to play with for this example at `weblogs.asp.net`.)

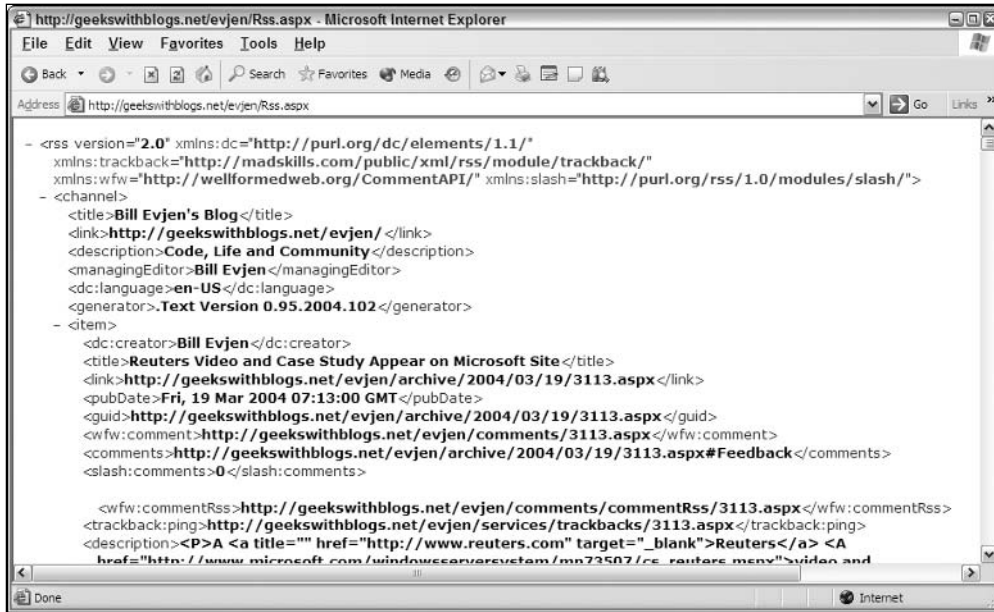


Figure 4-28

Now that you know the location of the XML from the blog, you can use this XML with the XmlDataSource control and display some of the results in a DataList control. The code for this example is shown in Listing 4-24.

Listing 4-24: Working with an RSS feed

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>XmlDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:DataList ID="DataList1" Runat="server" DataSourceID="XmlDataSource1">
            <HeaderTemplate>
                <table border="1" cellpadding="3">
            </HeaderTemplate>
            <ItemTemplate>
                <tr><td><b><%# XPath("title") %></b><br />
                <i><%# XPath("pubDate") %></i><br />
                <%# XPath("description") %></td></tr>
            </ItemTemplate>
            <AlternatingItemTemplate>
                <tr bgcolor="LightGrey"><td><b><%# XPath("title") %></b><br />
                <i><%# XPath("pubDate") %></i><br />
                <%# XPath("description") %></td></tr>
            </AlternatingItemTemplate>
        </asp:DataList>
    </form>
</body>
</html>
```

```

        </AlternatingItemTemplate>

        <FooterTemplate>
            </table>
        </FooterTemplate>
    </asp:DataList>

    <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
        DataFile="http://geekswithblogs.net/evjen/Rss.aspx"
        XPath="rss/channel/item">
    </asp:XmlDataSource>
</form>
</body>
</html>

```

Looking at the code in Listing 4-24, you can see that the `DataFile` points to a URL where the XML is retrieved. The `XPath` property filters out all the `<item>` elements from the RSS feed. The `DataList` control creates an HTML table and pulls out specific data elements from the RSS feed, such as the `<title>`, `<pubDate>`, and `<description>` elements.

Running this page in the browser, you get something similar to the results shown in Figure 4-29.

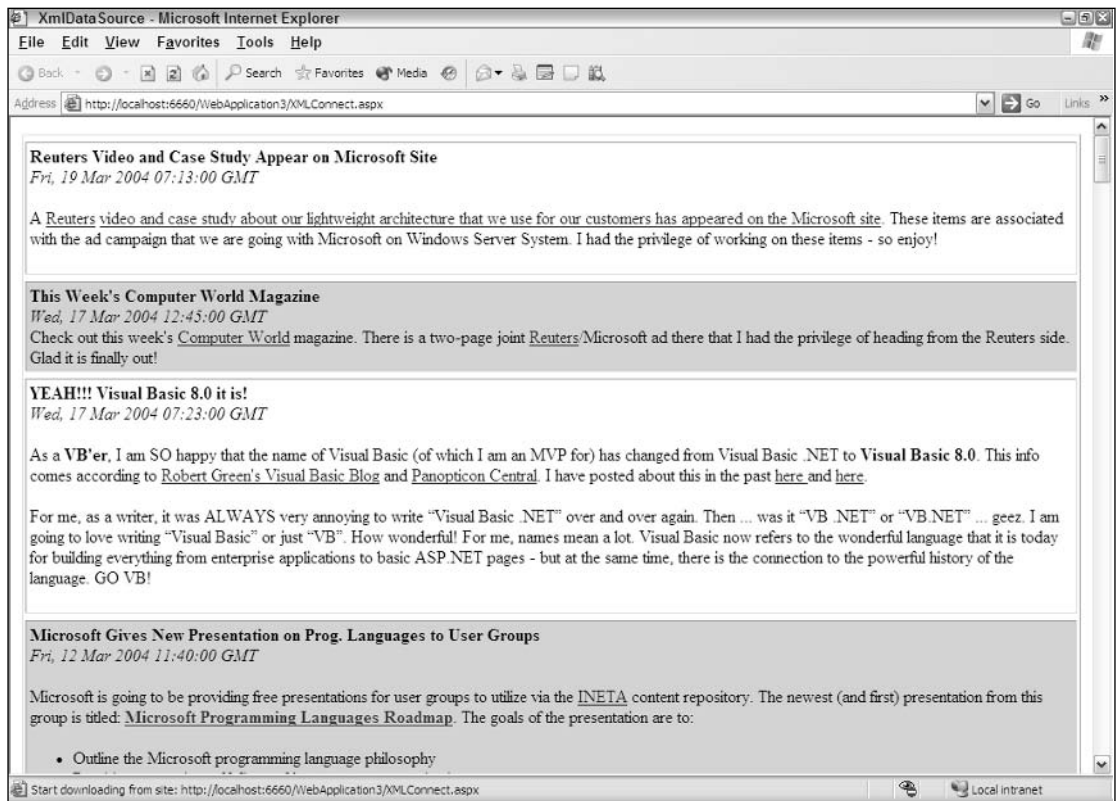


Figure 4-29

This approach also works with XML Web services, even ones for which you can pass in parameters using HTTP-GET. You just set up the `DataFile` value in the following manner:

```
DataFile="http://www.someserver.com/GetWeather.aspx/ZipWeather?zipcode=63301"
```

ObjectDataSource Control

The data source controls I have presented so far are really intended for applications working in a two-tiered environment. In these cases, the presentation piece works with a data store that might reside on another server. The `ObjectDataSource` control enables you to use data source controls that work in a three-tiered environment. This data source control interacts with an object that you have established, and that object then interacts with a data store elsewhere.

The object needs to be structured properly in order to work with the `ObjectDataSource` control, meaning that the object can perform basic operations such as selects, updates, inserts, and deletes.

For a simple example of working with the `ObjectDataSource` control, you can create a class that gets its data from another XML file called `Employee.xml`. The `Employee.xml` file is shown in Listing 4-25.

Listing 4-25: Employee.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Staff>
  <Employee>
    <FullName>Bill Evjen</FullName>
    <StartDate>02/2004</StartDate>
    <Salary>25,000</Salary>
  </Employee>
  <Employee>
    <FullName>Fred Cotterell</FullName>
    <StartDate>01/2000</StartDate>
    <Salary>32,000</Salary>
  </Employee>
  <Employee>
    <FullName>Tuija Pitkanen</FullName>
    <StartDate>04/2004</StartDate>
    <Salary>31,000</Salary>
  </Employee>
</Staff>
```

The `ObjectDataSource` control then works with a business object that returns the information from the XML file. For this, create a class called `Employees.vb` or `Employees.cs`. This class utilizes only the selected aspect of the object because you are only interested in reading the data. This class is shown in Listing 4-26.

Listing 4-26: Employees.vb/.cs

```
VB
Imports System.Data

Public Class Employees
```

```
Public Function GetEmployeeDetails() As DataSet
    Dim ds As New DataSet()
    ds.ReadXml(HttpContext.Current.Server.MapPath("Employee.xml"))

    Return ds
End Function
End Class
```

C#

```
using System;
using System.Web;
using System.Data;

public class Employees
{
    public DataSet GetEmployeeDetails()
    {
        DataSet ds = new DataSet();
        ds.ReadXml(HttpContext.Current.Server.MapPath("~/Employee.xml"));

        return ds;
    }
}
```

In this object, the `GetEmployeeDetails` method is the select statement utilized by the `ObjectDataSource` control in the ASP.NET page. This method returns its result using the types of `IEnumerable`, `DataSet`, or `DataTable`. In this example, it returns a `DataSet`. Now that the `Employees` class is in place and it returns a list of employees from the XML file, you can create an .aspx page that makes use of this object via an `ObjectDataSource` control.

The final step is to create an .aspx page that displays the results that come from the `Employees` class. This is illustrated in Listing 4-27.

Listing 4-27: Using the ObjectDataSource control

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>ObjectDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:GridView ID="GridView1" Runat="server"
            DataSourceID="ObjectDataSource1" BorderWidth="1px" BackColor="#DEBA84"
            CellPadding="3" CellSpacing="2" BorderStyle="None" BorderColor="#DEBA84">
            <FooterStyle ForeColor="#8C4510" BackColor="#F7DFB5">
            </FooterStyle>
            <PagerStyle HorizontalAlign="Center" ForeColor="#8C4510">
            </PagerStyle>
            <HeaderStyle ForeColor="White" Font-Bold="True" BackColor="#A55129">
            </HeaderStyle>
            <SelectedRowStyle ForeColor="White" Font-Bold="True"
                BackColor="#738A9C">
            </SelectedRowStyle>
```

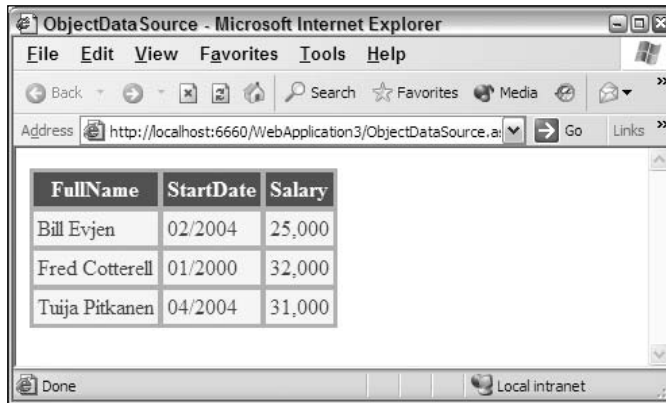
(continued)

Listing 4-27: (continued)

```
        </SelectedRowStyle>
        <RowStyle ForeColor="#8C4510" BackColor="#FFF7E7">
        </RowStyle>
    </asp:GridView>
    <asp:ObjectDataSource ID="ObjectDataSource1" Runat="server"
        TypeName="Employees" SelectMethod="GetEmployeeDetails">
    </asp:ObjectDataSource>
</form>
</body>
</html>
```

Look first at the `<asp:ObjectDataSource>` control. Only two attributes within this control are doing all the work. The first is the `TypeName` property. The value used with this property points to the name of the class with which it works. In this case, it is working with the `Employees` class contained in the `Employees.vb` or `Employees.cs` file. The second important property is the `SelectMethod` property. This property points to the select function contained in the `Employees` class. In this case, it is the only method that the `Employees` class has — the `GetEmployeeDetails` method. After these items are in place, you can bind this control to any of the data-bound controls. In this case, you use the `GridView` control and associate the two controls with the use of the `DataSourceID` property in the `GridView` control.

Putting all this together produces the results shown in Figure 4-30.



FullName	StartDate	Salary
Bill Evjen	02/2004	25,000
Fred Cotterell	01/2000	32,000
Tuija Pitkanen	04/2004	31,000

Figure 4-30

SiteMapDataSource Control

The `SiteMapDataSource` control is another new data source control that enables you to easily bind an application's site navigation hierarchy data to some of ASP.NET 2.0's latest navigation controls. In most cases, you bind the `SiteMapDataSource` control to an XML file that is referred to as a site map and has a `.sitemap` extension. These site maps define the navigation hierarchy of your application and can then be utilized by controls such as the `TreeView` control.

The `SiteMapDataSource` control and the corresponding navigation controls are discussed in detail in Chapter 5.

DataSetDataSource Control

The DataSetDataSource control enables you to bind ASP.NET controls to .NET-style DataSets. These might be DataSets that are produced from XML files, or from in-memory DataSets that you have created in your code. Using the DataSetDataSource is pretty straightforward because it is quite similar to working with the other data source controls. Listing 4-28 shows an example of using the DataSetDataSourceControl with the same Painters.xml file that was an example earlier in the chapter and binding it to a DropDownList control.

Listing 4-28: Using the DataSetDataSource control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>DataSetDataSource</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:DropDownList ID="DropDownlist1" Runat="server" DataTextField="name"
      DataSourceID="DataSetDataSource1">
    </asp:DropDownList>

    <asp:DataSetDataSource ID="DataSetDataSource1" Runat="server"
      DataFile="~/Painters.xml">
    </asp:DataSetDataSource>
  </form>
</body>
</html>
```

In this example, the DataSetDataSource control uses the DataFile attribute to get at the XML file, which is converted to a DataSet by the DataSetDataSource control. After it is converted, the DataSet can be used by the DropDownList control. The DropDownList control associates itself to the DataSetDataSource control via the use of the DataSourceID attribute. The field chosen to be displayed in the drop-down list is determined by using the DataTextField attribute. In this case, it points to the name attribute of the <Painters> element in the XML file. Running this page produces the results shown in Figure 4-31.

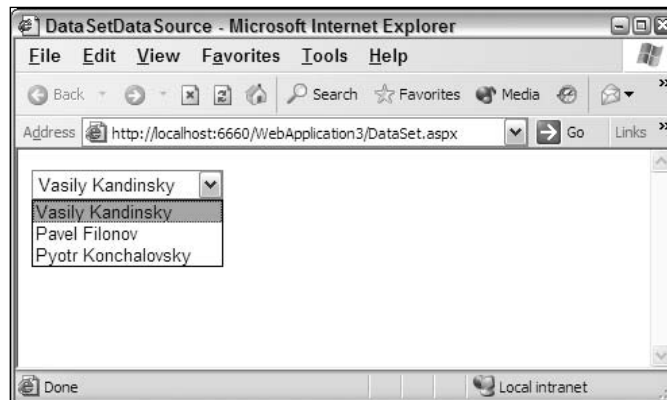


Figure 4-31

Visual Studio 2005

Visual Studio 2005 makes it rather easy and straightforward to create and configure data source controls and then bind them to any of the data-bound controls. So far in this chapter, you have looked at the creation of data source controls via the code that you write. This is the best way to learn about the capabilities of ASP.NET 2.0's data controls. You can also take the approach of simply dragging and dropping these controls onto the Visual Studio 2005 design surface and manipulating the controls in the IDE's Properties Window.

To create your data source controls and bind them to the data-bound controls, you can use the controls' smart tags. For an example of this, drag and drop a GridView control onto an empty design surface. Then below the GridView control, drag and drop a SqlDataSource control onto the page.

The first step is to set up your SqlDataSource control to get the data you want. To do this in the visual designer, you simply highlight the gray box that represents the SqlDataSource control and click the arrow in the upper-right-hand corner of the control. Doing this opens the control's smart tag and provides you with the option to configure and modify the control. The smart tag for the SqlDataSource control appears as shown in Figure 4-32.

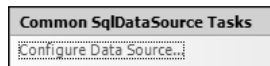


Figure 4-32

Within the smart tag for the SqlDataSource control, click the Configure Data Source link. This initiates the Data Source Configuration Wizard. The first step in the configuration process is to select the database you are connecting to. The wizard then shows you the connection string it's creating for you in a grayed-out pane (see Figure 4-33).

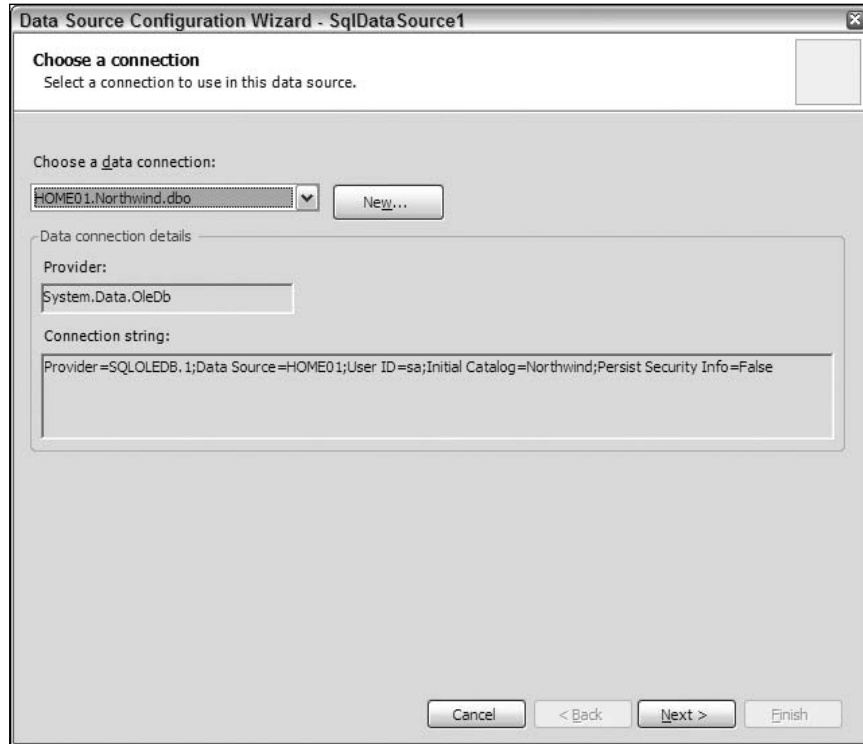


Figure 4-33

When you click the Next button, the Data Source Configuration Wizard gives you the option of storing the connection string in the `web.config` file, which is an outstanding idea because it is a safe way to store it. Any `.config` file is not browser-accessible and gives you a single place in the application where you can change the connection string and have that change take effect throughout the application. Keep the check box for this checked if you want to store the connection string in this manner (see Figure 4-34); otherwise, uncheck the box.

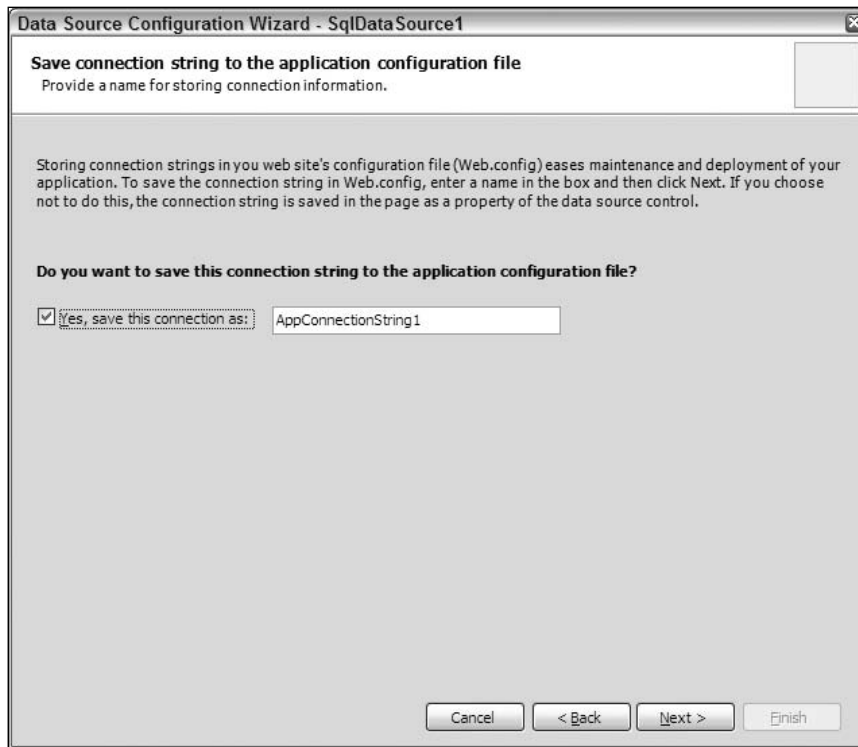


Figure 4-34

Again, click the Next button and configure the `Select` statement that the `SqlDataSource` control utilizes. Simply select the Customers table from the drop-down list and then check the * check box (see Figure 4-35). This means that you want each and every field from the Customers table.

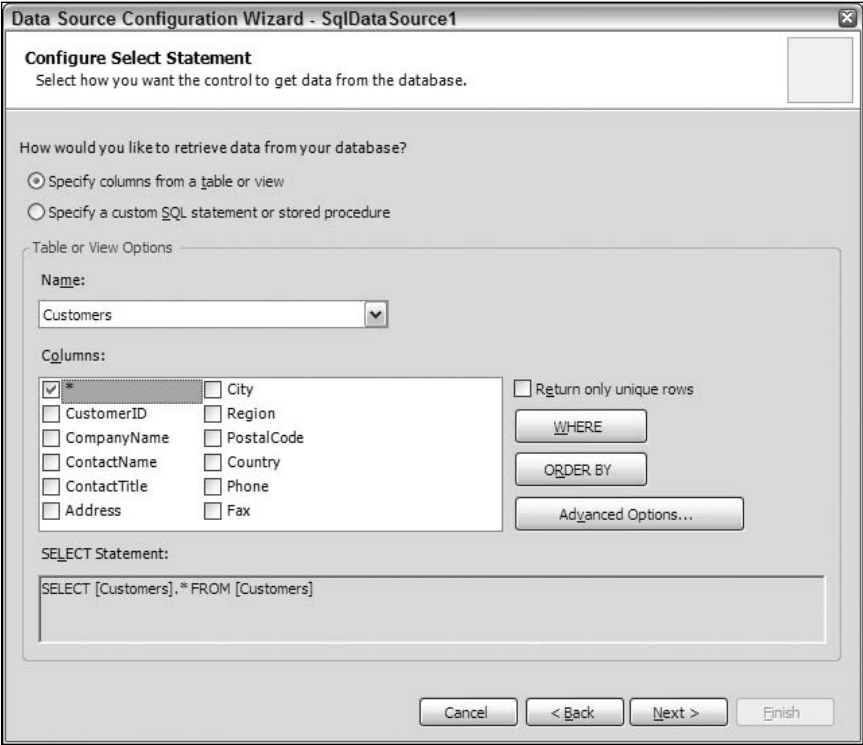


Figure 4-35

Click Next, and you can test out the SqlDataSource and its connection to the data store by clicking the Test Query button shown in Figure 4-36.

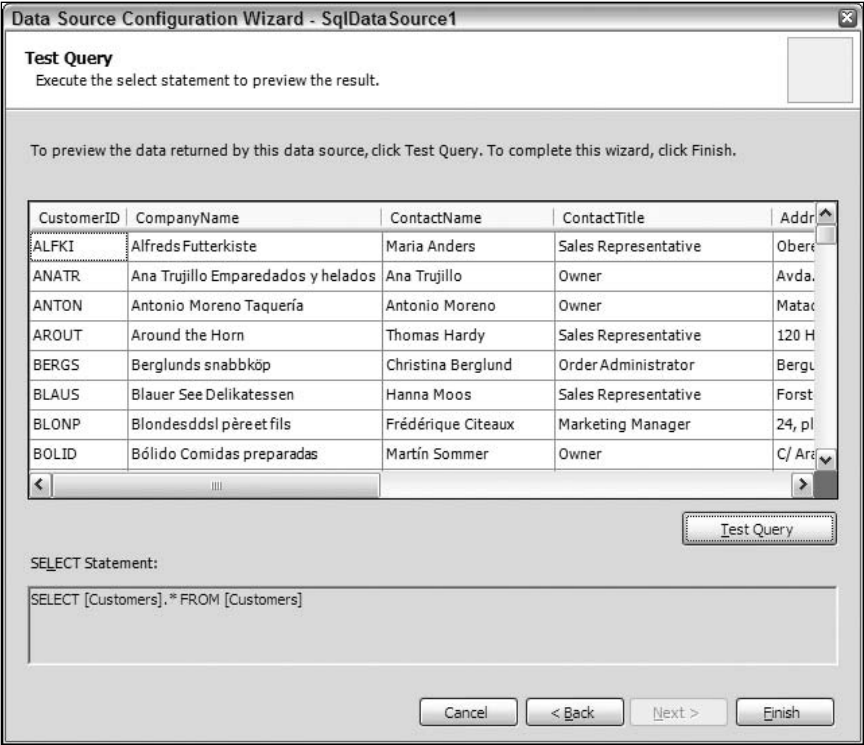


Figure 4-36

Clicking the Finish button brings you back to the design surface in Visual Studio. The SqlDataSource control is now ready to use with the GridView control. In the GridView control, from the drop-down list in the smart tag, select the SqlDataSource1 control and check the Enable Paging check box (see Figure 4-37). Next, you can create a better look and feel by choosing the Auto Format option.

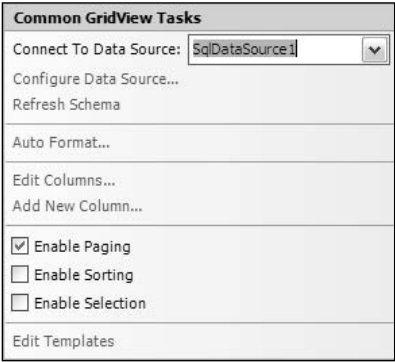


Figure 4-37

That’s it! Everything is in place and no coding is needed. Simply press F5 to run the page, and you get the results shown in Figure 4-38.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076545
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68306	Germany	0621-08460	0621-08924
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille		13008	France	91.24.45.40	91.24.45.41
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen BC		T2F 8M4	Canada	(604) 555-4729	(604) 555-3745

Figure 4-38

Connection Strings

An interesting point about the previous example is that we decided to store the connection string directly in the `web.config` file instead of hard-coding the connection string directly in the code of the page. This is always a good idea because it is more secure and makes your connection strings easier to manage. If you followed the previous steps as instructed, Visual Studio placed the connection string in the `web.config` for you. It then appears as shown in Listing 4-29.

Listing 4-29: Storing the connection string in the `web.config`

```
<?xml version="1.0"?>
<configuration>

  <connectionStrings>
    <add name="AppConnectionString1" connectionString="Provider=SQLOLEDB.1;
      Data Source=HOME01;User ID=sa;Initial Catalog=Northwind;
      Persist Security Info=False"
      providerName="System.Data.OleDb" />
  </connectionStrings>
</configuration>
```

(continued)

Listing 4-29: *(continued)*

```
</connectionStrings>

<system.web>
    ...
</system.web>

</configuration>
```

A `<connectionStrings>` section is created in the `web.config` file. Placed within the `<connectionStrings>` section is a simple `<add>` element that contains a couple of attributes. The first attribute is the `name` attribute. This is the name that is used to uniquely identify this connection string in your ASP.NET pages because it is possible to have multiple connection strings within your `web.config` file. In this case, the connection string is named `AppConnectionString1`.

The next attribute is the `connectionString` attribute. The value given here is the full connection string. The last attribute is the `providerName` attribute, which simply contains the name of the provider that you are using to connect to the data store.

Looking at the `SqlDataSource` code in the ASP.NET page utilizing this stored connection string in Listing 4-30, you can see that it is rather simple to use the information stored in the `web.config` file.

Listing 4-30: Using the connection string stored in the web.config file

```
<asp:SqlDataSource ID="SqlDataSource1" Runat="server"
    SelectCommand="SELECT [Customers].* FROM [Customers]"
    ConnectionString="<%$ ConnectionStrings:AppConnectionString1 %>"
    ProviderName="<%$ ConnectionStrings:AppConnectionString1.providername %>"
</asp:SqlDataSource>
```

You can see here that the connection string value is retrieved using `<%$ ConnectionStrings:AppConnectionString1 %>` and that the provider name attribute is accessed using `<%$ ConnectionStrings:AppConnectionString1.providername %>`.

Although this was done automatically for you by Visual Studio, you can just as easily do this yourself within the ASP.NET applications that you code.

Summary

This chapter introduced some of the new ways in which you can work with data in ASP.NET 2.0. The latest version of ASP.NET provides you with an outstanding new collection of data source controls that you can use to retrieve and manipulate data held in a wide variety of different data stores.

The new data source controls, `SqlDataSource`, `AccessDataSource`, `XmlDataSource`, `ObjectDataSource`, `DataSetDataSource`, and `SiteMapDataSource`, are powerful and easy to use. They require little effort on the part of the developer, but at the same time, they are rather extensible and can be modified for almost any purpose when it comes to working with the data that is running your applications.

In addition to the data source controls, this chapter took a look at some of the new data-bound server controls such as the `GridView` and the `DetailsView` controls. These are outstanding new controls that you can use with the new data source controls. The `GridView` is an enhanced `DataGrid` that has built-in paging and sorting, whereas the `DetailsView` control enables you to drill down into a particular data piece in a logical manner.

5

Site Navigation

The Web applications you develop generally have more than a single page. Usually you create a number of pages that are all interconnected in some fashion. If you also build the navigation around your pages, you make it easy for the end user to successfully work through your application in a straightforward manner.

Currently, you must choose among a number of different ways to expose to the end user the paths through your application. The difficult task of site navigation is compounded when you continue to add pages to the overall application.

The present method for building navigation within Web applications is to sprinkle pages with hyperlinks. Hyperlinks are generally added to Web pages by using include files or user controls. They can also be directly hard-coded onto a page so that they appear in the header or the side bar of the page being viewed. The difficulties in working with navigation become worse when you move pages around or change page names. Sometimes developers are forced to go to each and every page in the application just to change some aspect of the navigation.

ASP.NET 2.0 tackles this problem with the introduction of a navigation system that makes it quite trivial to manage how end users work through the applications you create. This new capability in ASP.NET is complex; but the great thing is that it can be as simple as you need it to be, or you can actually get in deep and control every aspect of how it works.

The new site navigation system includes the capability to define your entire site in an XML file, which is called a *site map*. After you define a new site map, a `SiteMap` class gives you the capability to programmatically work with it. Another addition in ASP.NET 2.0 is a new data provider that is specifically developed to work with site map files and to bind them to a new series of navigation-based server controls. This chapter takes a look at all these components in the new ASP.NET 2.0 navigation system. You can begin by looking at site maps.

Site Maps

Although a site map is not a required element (as you see later), one of the common first steps you take in working with the new ASP.NET 2.0 navigation system is building a site map for your application. A site map is an XML description of your site's structure.

You use this site map to define the layout of all the pages in your application and how they relate to one another. If you do this according to the new site map standard, you can interact with this navigation information using either the new `SiteMap` class or the new `SiteMapDataSource` control. By using the `SiteMapDataSource` control, you can bind the information in the site map file to databinding controls, including the new navigation server controls provided by ASP.NET 2.0.

To create a new site map file for your application, add an XML file to your application. When asked, you name the XML file `web.sitemap`. The file is named `app` and has the new file extension of `.sitemap`. Take a look at an example `.sitemap` file in Listing 5-1.

Listing 5-1: An example of a `web.sitemap` file

```
<?xml version="1.0" encoding="utf-8" ?>

<siteMap>
  <siteMapNode title="Home" description="Home Page" url="default.aspx">
    <siteMapNode title="News" description="The Latest News" url="News.aspx">
      <siteMapNode title="U.S." description="U.S. News"
        url="News.aspx?cat=us" />
      <siteMapNode title="World" description="World News"
        url="News.aspx?cat=world" />
      <siteMapNode title="Technology" description="Technology News"
        url="News.aspx?cat=tech" />
      <siteMapNode title="Sports" description="Sports News"
        url="News.aspx?cat=sport" />
    </siteMapNode>
    <siteMapNode title="Finance" description="The Latest Financial Information"
      url="Finance.aspx">
      <siteMapNode title="Quotes" description="Get the Latest Quotes"
        url="Quotes.aspx" />
      <siteMapNode title="Markets" description="The Latest Market Information"
        url="Markets.aspx">
        <siteMapNode title="U.S. Market Report"
          description="Looking at the U.S. Market" url="MarketsUS.aspx" />
        <siteMapNode title="NYSE"
          description="The New York Stock Exchange" url="NYSE.aspx" />
      </siteMapNode>
      <siteMapNode title="Funds" description="Mutual Funds"
        url="Funds.aspx" />
    </siteMapNode>
    <siteMapNode title="Weather" description="The Latest Weather"
      url="Weather.aspx" />
  </siteMapNode>
</siteMap>
```

So what does this file give you? Well, it gives you a logical structure that ASP.NET 2.0 can now use in the rest of the navigation system it provides. Next, examine how this file is constructed.

The root node of this XML file is a `<siteMap>` element. Only one `<siteMap>` element can exist in the file. Within the `<siteMap>` element, there is a `<siteMapNode>` element. This is generally the start page of the application. In the case of the file in Listing 5-1, the root `<siteMapNode>` points to the `default.aspx` page, the start page:

```
<siteMapNode title="Home" description="Home Page" url="default.aspx">
```

The following table describes each of the available attributes in the `<siteMapNode>` element.

Attribute	Description
Title	The title attribute provides a textual description of the link. The string value used here is the text used for the link.
Description	The description attribute not only reminds you what the link is for but is also used for the <code>ToolTip</code> attribute on the link. The <code>ToolTip</code> attribute is the yellow box that shows up next to the link when the end user hovers the cursor over the link for a couple of seconds.
url	The url attribute describes where the file is located in the solution. If the file is in the root directory, simply use the filename, such as <code>"default.aspx"</code> . If the file is located in a subfolder, be sure to include the folders in the string value used in this attribute. For example, <code>"MySubFolder/Markets.aspx"</code> .

After you have the first `<siteMapNode>` in place, you can place as many additional `<siteMapNode>` elements as you need. You can also create additional link-levels by creating child `<siteMapNode>` elements for any parent `<siteMapNode>` in the structure.

The example in Listing 5-1 gives the application the following navigation structure.

```

Home
  News
    U.S.
    World
    Technology
    Sports
  Finance
    Quotes
    Markets
      U.S. Market Report
      NYSE
    Funds
  Weather

```

From this structure, you can see that it goes down three levels in some places. One of the easiest places to use this file is with the new `SiteMapPath` server control that now comes with ASP.NET 2.0. Take a close look at this new server control.

SiteMapPath Server Control

It is quite easy to use the `.sitemap` file you just created with the new SiteMapPath server control provided with ASP.NET 2.0. You find this new control in the Navigation section of the Visual Studio 2005 IDE.

The SiteMapPath control creates navigation functionality that you have either created yourself or have seen elsewhere in Web pages on the Internet. The SiteMapPath control creates what some refer to as *breadcrumb navigation*. This is a linear path view of where the end user is in the navigation structure. The Reuters.com Web site, shown in Figure 5-1, uses this type of navigation. A black arrow points out the breadcrumb navigation used on the page.

The purpose of this type of navigation is to show end users where they are in relation to the rest of the site. Traditionally, coding this kind of navigation has been tricky, to say the least; but now with the introduction of the SiteMapPath server control, you should find coding for this type of navigation a breeze.

You should first create an application that has the `web.sitemap` file created in Listing 5-1. From there, create a WebForm called `MarketsUS.aspx`. This is a file defined in the `web.sitemap` file to be on the lowest tier of files in the application.

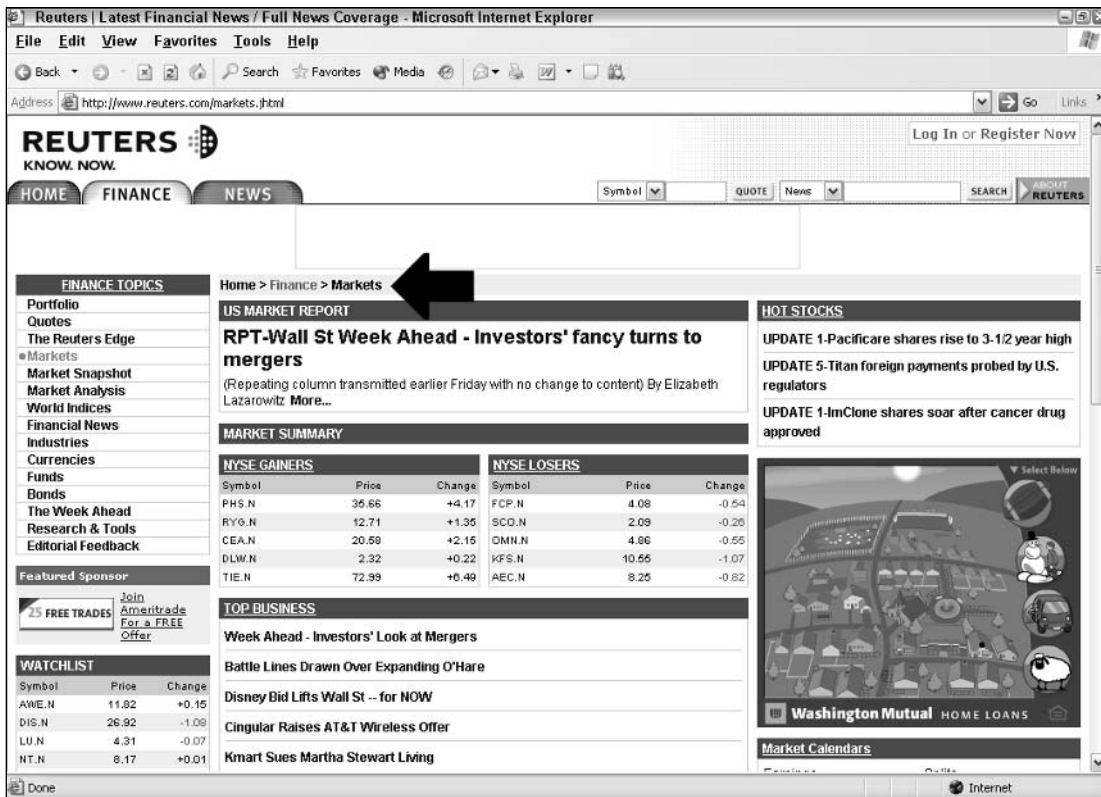


Figure 5-1

The SiteMapPath control is so easy to work with that it doesn't even require a datasource control to hook it up to the `web.sitemap` file where it infers all of its information. All you do is drag and drop a SiteMapPath control onto your `MarketsUS.aspx` page. In the end, you should have a page like the one shown in Listing 5-2.

Listing 5-2: Using the web.sitemap file with a SiteMapPath server control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head ID="Head1" runat="server">
  <title>Using the SiteMapPath Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:SiteMapPath ID="Sitemappath1" Runat="server">
    </asp:SiteMapPath>
  </form>
</body>
</html>
```

Not much to it, is there? It really is that easy. Run this page and you see the results shown in Figure 5-2.

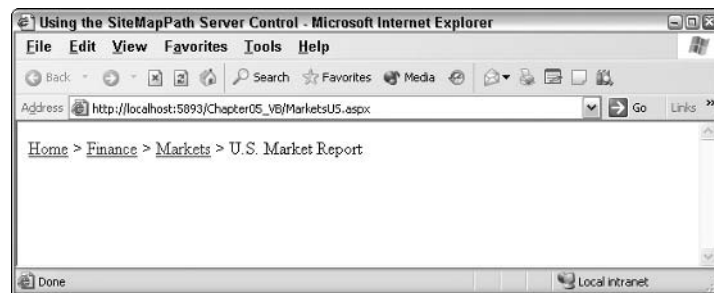


Figure 5-2

This screen shot shows that you are on the U.S. Market Report page at `MarketsUS.aspx`. As an end user, you can see that this page is part of the Markets section of the site, which, in turn, is part of the Finance section of the site. With breadcrumb navigation, end users, understanding the structure of the site and their place in it, can quickly click the links to navigate to where they want to go in the site.

If you hover your mouse over the Finance link, you see a tooltip appear after a couple of seconds, as shown in Figure 5-3.

This tooltip, which reads `The Latest Financial Information`, comes from the description attribute of the `<siteMapNode>` element in the `web.sitemap` file.

```
<siteMapNode title="Finance" description="The Latest Financial Information"
  url="Finance.aspx">
```



Figure 5-3

The SiteMapPath control works automatically by itself with very little work on your part. You just add the basic control to your page, and the breadcrumb navigation you have just seen is automatically created. However, you can use the properties discussed in the following sections to modify the control's appearance and behavior.

The PathSeparator property

One important property for the SiteMapPath control is the PathSeparator property. By default, the SiteMapPath control uses a greater than sign (>) to separate the link elements. You can change this by reassigning a new value to the PathSeparator property. Listing 5-3 illustrates the use of this property.

Listing 5-3: Changing the PathSeparator value

```
<asp:SiteMapPath ID="Sitemappath1" Runat="server" PathSeparator=" | ">
</asp:SiteMapPath>
```

Or

```
<asp:SiteMapPath ID="Sitemappath1" Runat="server">
  <PathSeparatorTemplate> | </PathSeparatorTemplate>
</asp:SiteMapPath>
```

The SiteMapPath control in this example uses the pipe character (|), which is found above your Enter key. When it is rendered, you get the results shown in Figure 5-4.

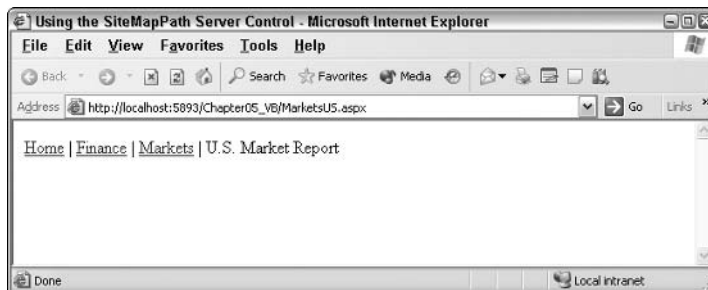


Figure 5-4

As you can see, you can use either the `PathSeparator` property or the `<PathSeparatorTemplate>` element within the `SiteMapPath` control.

With the use of the `PathSeparator` property or the `<PathSeparatorTemplate>` element, it is quite easy to specify what you want to use to separate the links in the breadcrumb navigation, but you might also want to give this pipe some visual style as well. You can add a `<PathSeparatorStyle>` node to your `SiteMapPath` control. An example of this is shown in Listing 5-4.

Listing 5-4: Adding style to the `PathSeparator` property

```
<asp:SiteMapPath ID="Sitemappath1" Runat="server" PathSeparator=" | ">
  <PathSeparatorStyle Font-Bold="true" Font-Names="Verdana" ForeColor="#663333"
    BackColor="#cccc66"></PathSeparatorStyle>
</asp:SiteMapPath>
```

Okay, it may not be pretty (I am not much of a designer), but by using the `<PathSeparatorStyle>` element with the `SiteMapPath` control, I am able to change the visual appearance of the separator elements. The results are shown in Figure 5-5.

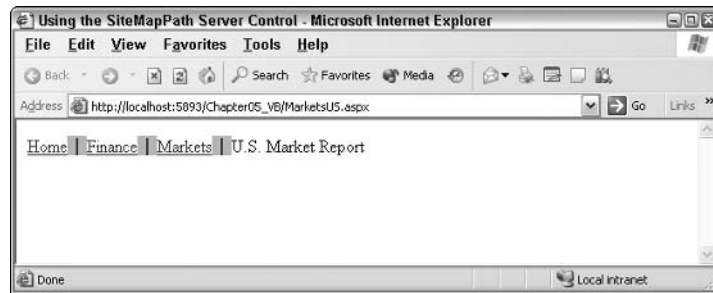


Figure 5-5

Using these constructs, you can also add an image as the separator, as illustrated in Listing 5-5.

Listing 5-5: Using an image as the separator

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head ID="Head1" runat="server">
  <title>Using the SiteMapPath Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:SiteMapPath ID="Sitemappath1" Runat="server">
      <PathSeparatorTemplate>
        <asp:Image ID="Image1" Runat="server" ImageUrl="divider.gif" />
      </PathSeparatorTemplate>
    </asp:SiteMapPath>
  </form>
</body>
</html>
```

To utilize an image as the separator between the links, you use the `<PathSeparatorTemplate>` element and place an Image control within it. In fact, you can choose any type of control you want to be placed between the navigation links that the SiteMapPath control produces.

The PathDirection property

Another interesting property to use with the SiteMapPath control is `PathDirection`. This property changes the direction of the links generated in the output. Only two settings are possible for this property: `RootToCurrent` and `CurrentToRoot`.

The Root link is the first link in the display. This is usually the Home page. The Current link is the link for the page currently being displayed. By default, this property is set to `RootToCurrent`. Changing the example to `CurrentToRoot` produces the results shown in Figure 5-6.

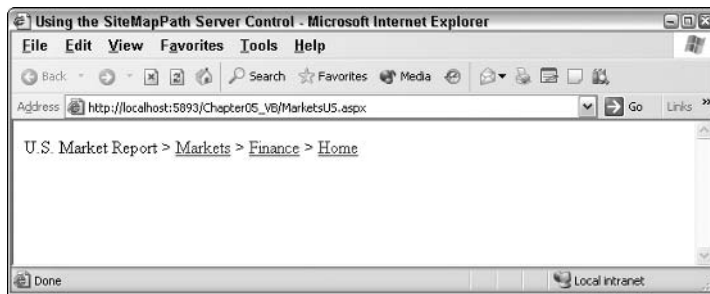


Figure 5-6

The ParentLevelsDisplayed property

In some cases, your navigation may go quite deep. You can see on the site map shown in Listing 5-1 that you go three pages deep, which isn't a big deal. Some of you, however, might be dealing with sites that go quite a number of pages deeper. In these cases, it might be bit silly to use the SiteMapPath control. Doing so would display a huge list of pages.

In a case like this, you can turn to the `ParentLevelsDisplayed` property that is part of the SiteMapPath control. When set, this property displays pages only as deep as specified. Therefore, if you are using the SiteMapPath control with the `web.sitemap`, as shown in Listing 5-1, and you give the `ParentLevelsDisplayed` property a value of 3, you don't notice any change to your page. It already displays the path three pages deep. If you change this value to 2, however, the SiteMapPath control is constructed as follows:

```
<asp:SiteMapPath ID="Sitemappath1" Runat="server" ParentLevelsDisplayed="2">
</asp:SiteMapPath>
```

Notice the result of this change in Figure 5-7. The SiteMapPath control shows links only two pages deep and doesn't show the Home page link.

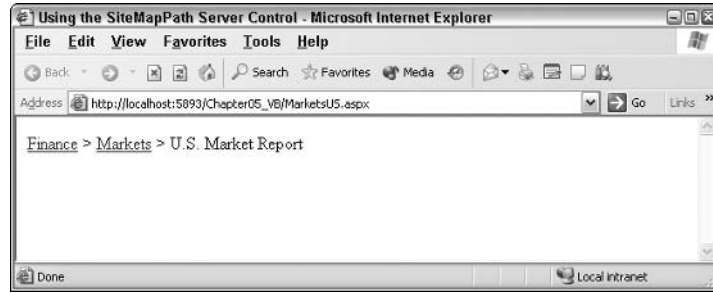


Figure 5-7

By default, no limit is set on the number of links shown, so the SiteMapPath control just generates the specified number of links based on what is labeled in the site map file.

The ShowToolTips property

By default, the SiteMapPath control generates tooltips for each link if a description property is used within the web.sitemap file. Remember, a tooltip is the text that appears on-screen when an end user hovers the mouse over one of the links in the SiteMapPath control. I showed you this capability earlier in this chapter.

There may be times when you do not want your SiteMapPath control to show any tooltips for the links that it generates. For these moments, you can actually turn off this capability in a couple of ways. The first way is not to include any description attributes in the .sitemap file. If you remove these attributes from the file, the SiteMapPath has nothing to display for the tooltips on the page.

The other way to turn off the display of tooltips is to set the ShowToolTips property to `False`, as shown here:

```
<asp:SiteMapPath ID="Sitemappath1" Runat="server" ShowToolTips="false">
</asp:SiteMapPath>
```

This turns off the tooltips capability but still allows you to use the description property in the .sitemap file. You may still want to use the description attribute because it allows you to keep track of what the links in your file are used for. This is quite advantageous when you are dealing with hundreds or even thousands of links in your application.

The SiteMapPath control's child elements

You already saw the use of the `<PathSeparatorStyle>` and the `<PathSeparatorTemplate>` child elements for the SiteMapPath control, but additional child elements exist. The following table covers each of the available child elements.

Child Element	Description
CurrentNodeStyle	Applies styles to the link in the SiteMapPath navigation for the currently displayed page.
CurrentNodeTemplate	Applies a template construction to the link in the SiteMapPath navigation for the currently displayed page.
HoverNodeStyle	Applies styles to the link in the SiteMapPath navigation over which the end user is hovering the cursor.
NodeStyle	Applies styles to all links in the SiteMapPath navigation. The settings applied in the <code>CurrentNodeStyle</code> or <code>RootNodeStyle</code> elements supersede any settings placed here.
NodeStyleTemplate	Applies a template construction to all links in the SiteMapPath navigation. The settings applied in the <code>CurrentNodeStyle</code> or <code>RootNodeStyle</code> elements supersede any settings placed here.
PathSeparatorStyle	Applies styles to the link dividers in the SiteMapPath navigation.
PathSeparatorTemplate	Applies a template construction to the link dividers in the SiteMapPath navigation.
RootNodeStyle	Applies styles to the first link (the root link) in the SiteMapPath navigation.
RootNodeTemplate	Applies a template construction to the first link in the SiteMapPath navigation.

TreeView Server Control

I have to say that I really like this new control. The TreeView server control is a rich server control for rendering a hierarchy of data, so it is quite ideal for displaying what is contained in your `.sitemap` file. Figure 5-8 shows you how it displays the contents of the site map (from Listing 5-1) that you have been working with thus far in this chapter. This figure first shows a completely collapsed TreeView control at the top of the screen, while the second TreeView control has been completely expanded.

This control can preload the nodes to be displayed, even if they are hidden, at first, by the collapsible framework of the control. If you use a client-side script, the control does not need to make a call back to the server if someone expands one of the nodes in the control. Just the fact that it won't make a postback and redraw the page gives this control a snappiness that will cause your end users to really enjoy using it. Of course, this capability is only there if the browser accepts the client-side code that the TreeView control can generate. If not, the control knows this and renders only what is appropriate and performs postbacks for those clients who cannot work with this client-side script.

The TreeView control is quite customizable, but first take a look at how to create a default version of the control using the `.sitemap` file from Listing 5-1. For this example, continue to use the `MarketsUS.aspx` page you created earlier.

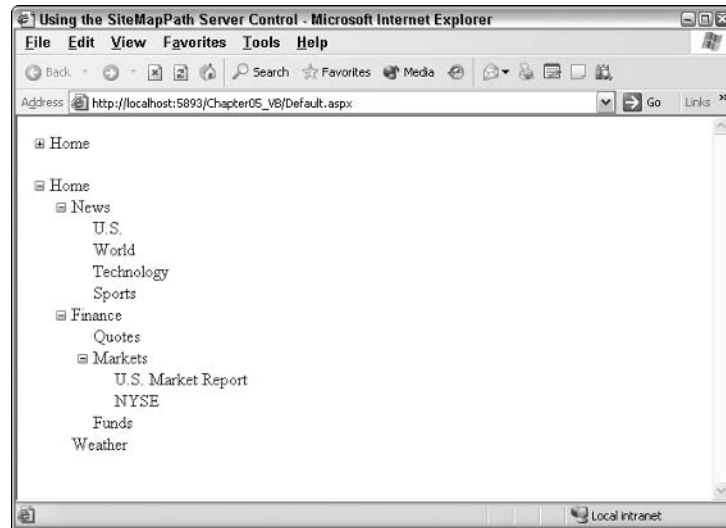


Figure 5-8

The first step is to create a SiteMapDataSource control on the page. When working with the TreeView control that displays the contents of your .sitemap file, you must apply one of these datasource controls. The TreeView control doesn't just bind to your site map file automatically as the SiteMapPath control does.

After a basic SiteMapDataSource control is in place, position a TreeView control on the page and set the DataSourceId property to SiteMapDataSource1. When you have finished, your code should look like Listing 5-6.

Listing 5-6: A basic TreeView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head ID="Head1" runat="server">
    <title>Using the SiteMapPath Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapPath ID="Sitemappath1" Runat="server">
        </asp:SiteMapPath>
        <br /><p>
        <asp:TreeView ID="Treeview1" Runat="server"
            DataSourceID="Sitemapdatasource1">
        </asp:TreeView>
        <asp:SiteMapDataSource ID="Sitemapdatasource1" Runat="server" /></p>
        </form>
    </body>
</html>
```

After the page is run and the TreeView control is expanded, the results are displayed as shown in Figure 5-9.

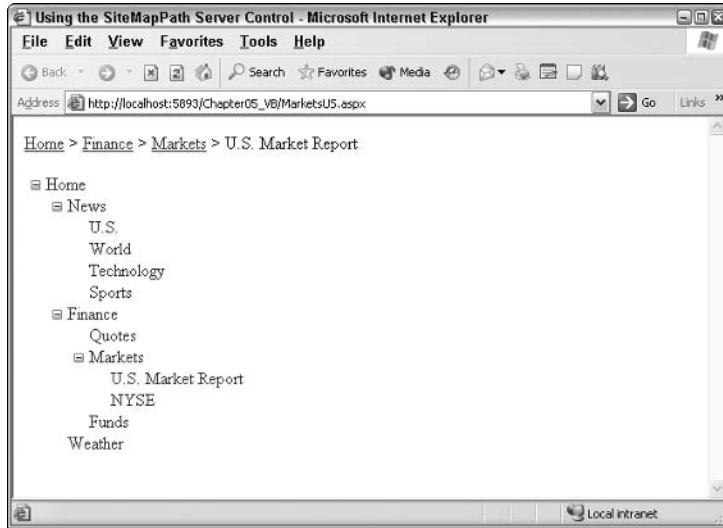


Figure 5-9

This is a very basic TreeView control. The great thing about this control is that it allows for a high degree of customization and even gives you the capability to use some predefined styles that come prepackaged with ASP.NET 2.0.

Identifying the TreeView control's built-in styles

As stated, the TreeView control does come with a number of pre-built styles right out of the box. The best way to utilize these predefined styles is to do so from the Design view of your page. By right-clicking on the TreeView control on your page from the Design view in Visual Studio 2005, you find the Auto Format option. Click this option and a number of styles become available to you. Selecting one of these styles changes the code of your TreeView control to adapt to that chosen style. For instance, if you choose MSDN from the list of options, the simple one-line TreeView control you created is converted to what is shown in Listing 5-7.

Listing 5-7: A TreeView control with the MSDN style applied to it

```
<asp:TreeView ID="Treeview1" Runat="server" DataSourceID="Sitemapdatasource1"
nodeindent="10" font-names="Verdana" font-size="8pt" forecolor="Black"
imageset="Msdn">
  <HoverNodeStyle BackColor="#CCCCCC" VerticalPadding="1" BorderColor="#888888"
  BorderStyle="Solid" BorderWidth="1px" Font-Underline="True"
  HorizontalPadding="3">
  </HoverNodeStyle>
  <SelectedNodeStyle BackColor="White" VerticalPadding="1" BorderColor="#888888"
  BorderStyle="Solid" BorderWidth="1px" HorizontalPadding="3">
  </SelectedNodeStyle>
  <NodeStyle VerticalPadding="2" Font-Names="Verdana" Font-Size="8pt"
  NodeSpacing="1" HorizontalPadding="5" ForeColor="Black">
  </NodeStyle>
</asp:TreeView>
```

As you can see, if you use these built-in styles, it isn't too difficult to completely change the look and feel of the TreeView control. When this bit of code is run, you get the results shown in Figure 5-10.

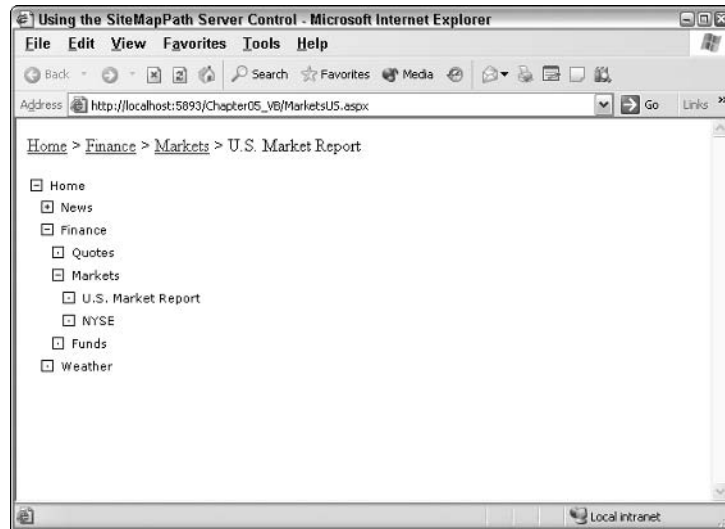


Figure 5-10

Examining the parts of the TreeView control

To master working with the TreeView control, you must understand the terminology used for each part of the hierarchical tree that is created by the control.

First, every element or entry in the TreeView control is called a *node*. The uppermost node in the hierarchy of nodes is the *root node*. It is possible for a TreeView control to have multiple root nodes. Any node, including the root node, is also considered a *parent node* if it has any nodes that are directly under it in the hierarchy of nodes. The nodes directly under this parent node are referred to as *child nodes*. Each parent node can have one or more child nodes. Finally, if a node contains no child nodes, it is referred to as a *leaf node*.

The following listing, based on the site map shown earlier, details the use of this terminology:

```
Home - Root node, parent node
  News - Parent node, child node
    U.S. - Child node, leaf node
    World - Child node, leaf node
    Technology - Child node, leaf node
    Sports - Child node, leaf node
  Finance - Parent node, child node
    Quotes - Child node, leaf node
    Markets - Parent node, child node
      U.S. Market Report - Child node, leaf node
      NYSE - Child node, leaf node
    Funds - Child node, leaf node
  Weather - Child node, leaf node
```

From this listing, you can see what each node is and how it is referred in the hierarchy of nodes. For instance, the `U.S. Market Report` node is a leaf node — meaning that it doesn't have any child nodes associated with it. However, it is also a child node to the `Markets` node, which is a parent node to the `U.S. Market Report` node. If you are working with the `Markets` node directly, it is also a child node to the `Finance` node, which is its parent node. The main point to take away from all this is that each node in the site map hierarchy has a relationship to the other nodes in the hierarchy. You must understand these relationships because you can programmatically work with these nodes (as will be demonstrated later in this chapter) and the methods used for working with them includes terms like `RootNode`, `CurrentNode` and `ParentNode`.

Binding the TreeView control to an XML file

You are not limited to working with just a `.sitemap` file in order to populate the nodes of your `TreeView` controls. You have many ways to get this done. One cool way is to use the `XmlDataSource` control (instead of the `SiteMapDataSource` control) to populate your `TreeView` controls from your XML files.

For an example of this, suppose that you created a hierarchical list of items in an XML file called `Hardware.xml`. An example of this is shown in Listing 5-8.

Listing 5-8: Hardware.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Hardware>
  <Item Category="Motherboards">
    <Option Choice="Asus" />
    <Option Choice="Abit" />
  </Item>
  <Item Category="Memory">
    <Option Choice="128mb" />
    <Option Choice="256mb" />
    <Option Choice="512mb" />
  </Item>
  <Item Category="HardDrives">
    <Option Choice="40GB" />
    <Option Choice="80GB" />
    <Option Choice="100GB" />
  </Item>
  <Item Category="Drives">
    <Option Choice="CD" />
    <Option Choice="DVD" />
    <Option Choice="DVD Burner" />
  </Item>
</Hardware>
```

As you can see, this list is not meant to be used for site navigation purposes, but instead for allowing the end user to make a selection from a hierarchical list of options. This XML file is divided into four categories of available options: `motherboards`, `memory`, `harddrives`, and `drives`. To bind your `TreeView` control to this XML file, use an `XmlDataSource` control that specifies the location of the XML file you are going to use. Then within the `TreeView` control itself, use the `<asp:TreeNodeBinding>` element to specify which elements to bind in the XML file to populate the nodes of the `TreeView` control. This is illustrated in Listing 5-9:

Listing 5-9: Binding a TreeView control to the Hardware.xml file

```

<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Latest Hardware</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TreeView ID="Treeview1" Runat="server" DataSourceID="Xmldatasource1">
            <DataBindings>
                <asp:TreeNodeBinding DataMember="Hardware"
                    Text="Computer Hardware" />
                <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
                <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
            </DataBindings>
        </asp:TreeView>
        <asp:XmlDataSource ID="Xmldatasource1" Runat="server"
            DataFile="Hardware.xml">
        </asp:XmlDataSource>
    </form>
</body>
</html>

```

The first item to look at is the `<asp:XmlDataSource>` control. It is just as simple as the previous `<asp:SiteMapDataSource>` control, but it points at the `Hardware.xml` file using the `DataFile` property.

The next step is to create a `TreeView` control that binds to this particular XML file. You can bind a default `TreeView` control directly to the `XmlDataSource` control like this:

```

<asp:TreeView ID="TreeView1" Runat="server"
    DataSourceID="XmlDataSource1" />

```

Doing this, you get the *incorrect* result shown in Figure 5-11.



Figure 5-11

As you can see, the TreeView control binds just fine to the `Hardware.xml` file, but looking at the nodes within the TreeView control, you can see that it is simply displaying the names of the actual XML elements from the file itself. Because this isn't what you want, you specify how to bind to the XML file with the use of the `<DataBindings>` element within the TreeView control.

The `<DataBindings>` element encapsulates one or more `TreeNodeBinding` objects. Two of the more important available properties of a `TreeNodeBinding` object are the `DataMember` and `TextField` properties. The `DataMember` property points to the name of the XML element that the TreeView control should look for. The `TextField` property specifies the XML attribute that the TreeView should look for in that particular XML element. If you do this correctly with the use of the `<DataBindings>` construct, you get the result shown in Figure 5-12.



Figure 5-12

You can also see from Listing 5-9 that you can override the text value of the root node from the XML file, `<Hardware>`, and have it appear as `Computer Hardware` in the TreeView control.

```
<asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
```

Selecting multiple options in a TreeView

As I stated earlier, the TreeView control is not meant to be used primarily for navigation purposes. Instead, you can use it for all sorts of things. In many cases, you can present a hierarchical list from which you want the end user to select one or more items.

One great built-in feature of the TreeView control is the capability to put check boxes next to nodes within the hierarchical items in the list. These boxes allow end users to make multiple selections. The TreeView control contains a property called `ShowCheckBoxes`, which can be used to create check boxes next to many different types of nodes within a list of items.

The available values for the `ShowCheckBoxes` property are discussed in the following table.

Value	Description
All	Applies check boxes to each and every node within the TreeView control.
Leaf	Applies check boxes to only the nodes that have no additional child elements.
None	Applies no check boxes to any node within the TreeView control.
Parent	Applies check boxes to only the nodes considered parent nodes within the TreeView control. A parent node has at least one child node associated with it.
Root	Applies a check box to any root node contained within the TreeView control.

When working with the `ShowCheckBoxes` property, you can set it declaratively in the control itself:

```
<asp:TreeView ID="Treeview1" Runat="server" Font-Underline="false"
  DataSourceID="Xmldatasource1" ShowCheckBoxes="leaf">
  ...
</asp:TreeViewTreeView>
```

Or you can set it programmatically by using the following code:

VB

```
TreeView1.ShowCheckBoxes = TreeNodeTypes.Leaf
```

C#

```
TreeView1.ShowCheckBoxes = TreeNodeTypes.Leaf;
```

For an example of using check boxes with the TreeView control, let's continue to expand on the computer hardware example from Listing 5-9. Create a hierarchical list that enables people to select multiple items from the list in order to receive additional information about the selected items. Listing 5-10 shows an example of this.

Listing 5-10: Applying check boxes next to the leaf nodes within the hierarchical list of nodes

VB

```
<%@ Page Language="VB" %>
```

```
<script runat="server">
  Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    If TreeView1.CheckedNodes.Count > 0 Then
      Label1.Text = "We are sending you information on:<p>"

      For Each node As TreeNode In TreeView1.CheckedNodes
        Label1.Text += node.Text & " " & node.Parent.Text & "<br>"
      Next
    Else
      Label1.Text = "You didn't select anything. Sorry!"
    End If
  End Sub
</script>
```

(continued)

Listing 5-10: (continued)

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Latest Hardware</title>
</head>
<body>
    <form runat="server">
        Please select the items you are interested in:
        <p>
            <asp:TreeView ID="Treeview1" Runat="server" Font-Underline="false"
                DataSourceID="Xmldatasource1" ShowCheckBoxes="leaf">
                <DataBindings>
                    <asp:TreeNodeBinding DataMember="Hardware"
                        Text="Computer Hardware" />
                    <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
                    <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
                </DataBindings>
            </asp:TreeView>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Submit Choices"
                OnClick="Button1_Click" />
        </p>
        <asp:XmlDataSource ID="XmlDataSource1" Runat="server"
            DataFile="Hardware.xml">
        </asp:XmlDataSource>
        <p>
            <asp:Label ID="Label1" Runat="Server" />
        </p>
    </form>
</body>
</html>
```

C#

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

```
<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        if (TreeView1.CheckedNodes.Count > 0)
        {
            Label1.Text = "We are sending you information on:<p>";
            foreach (TreeNode node in TreeView1.CheckedNodes)
            {
                Label1.Text += node.Text + " " + node.Parent.Text + "<br>";
            }
        }
        else
        {
            Label1.Text = "You didn't select anything. Sorry!";
        }
    }
</script>
```

In this example, you first set the `ShowTextBoxes` property to `Leaf`, meaning that you are only interested in having check boxes appear next to items in the `TreeView` control which do not contain any child nodes. The only items with check boxes next to them should be the last item that can be expanded in the hierarchical list.

After this property is set, you then work with the items that are selected by the end user in the `Button1_Click` event. The first thing you should check is whether any selection at all was made:

```
If TreeView1.CheckedNodes.Count > 0 Then
    ...
End If
```

In this case, the number of checked nodes on the postback needs to be greater than zero, meaning that at least one was selected. If so, you can execute the code within the `If` statement. The `If` statement then proceeds to populate the `Label` control that is on the page. To populate the `Label` control with data from the selected nodes, you use a `For Each` statement, as shown in the following:

```
For Each node As TreeNode In TreeView1.CheckedNodes
    ...
Next
```

This creates an instance of a `TreeNode` object and checks each `TreeNode` object within the `TreeView1` collection of checked nodes.

For each node that is checked, you grab the nodes `Text` value and the `Text` value of this node's parent node to further populate the `Label` control:

```
Label1.Text += node.Text & " " & node.Parent.Text & "<br>"
```

In the end, you get a page that produces the results shown in Figure 5-13.

Specifying custom icons in the TreeView control

The `TreeView` control allows for a high degree of customization. You saw earlier in the chapter that you were easily able to customize the look and feel of the `TreeView` control by specifying one of the built-in styles. Applying one of these styles dramatically changes the appearance of the control. One of the most noticeable changes concerns the icons used for the nodes within the `TreeView` control. Although it is not as easy as just selecting one of the styles built into the `TreeView` control, you can apply your own icons to be used for the nodes within the hierarchical list of nodes.

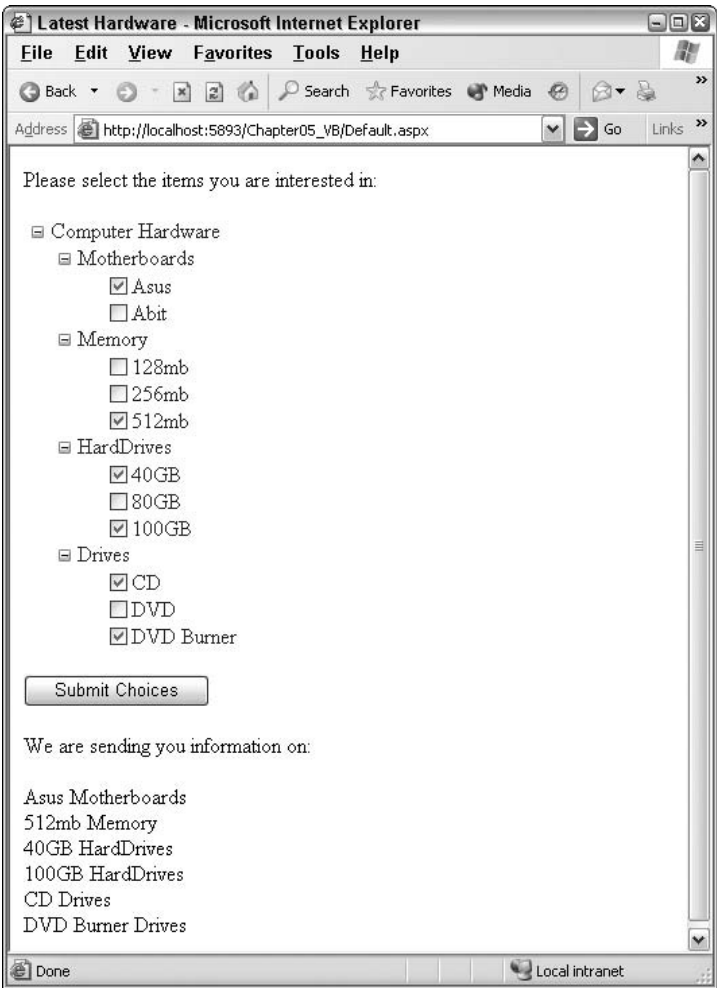


Figure 5-13

The TreeView control contains the properties discussed in the following table. These properties enable you to specify your own images to use for the nodes of the control.

Property	Description
CollapseImageUrl	Applies a custom image next to nodes that have been expanded to show any of their child nodes and have the capability of being collapsed.
ExpandImageUrl	Applies a custom image next to nodes that have the capability of being expanded to display their child nodes.
LeafImageUrl	Applies a custom image next to a node that has no child nodes and is last in the hierarchical chain of nodes.

Property	Description
NoExpandImageUrl	Applies a custom image to nodes that, for programmatic reasons, cannot be expanded or to nodes that are leaf nodes. This is primarily used for spacing purposes to align leaf nodes with their parent nodes.
ParentNodeImageUrl	Applies a custom image only to the parent nodes within the TreeView control.
RootNodeImageUrl	Applies a custom image next to only the root nodes within the TreeView control.

Listing 5-11 shows an example of these properties in use.

Listing 5-11: Applying custom images to the TreeView control

```
<asp:TreeView ID="TreeView1" Runat="server" Font-Underline="false"
  DataSourceId="XmlDataSource1"
  CollapseImageUrl="Images/CollapseImage.gif"
  ExpandImageUrl="Images/ExpandImage.gif"
  LeafImageUrl="Images/LeafImage.gif">
  <DataBindings>
    <asp:TreeNodeBinding DataMember="Hardware" Text="Computer Hardware" />
    <asp:TreeNodeBinding DataMember="Item" TextField="Category" />
    <asp:TreeNodeBinding DataMember="Option" TextField="Choice" />
  </DataBindings>
</asp:TreeView>
```

Specifying these three images to precede the nodes in your control overrides the default values of using a plus (+) sign and a minus (–) sign for the expandable and collapsible nodes. It also overrides simply using an image for any leaf nodes when by default nothing is used. Using the code from Listing 5-11, you get something similar to the results illustrated in Figure 5-14.

Specifying lines used to connect nodes

Because the TreeView control shows a hierarchical list of items to the end user, you sometimes want to show the relationship between these hierarchical items more explicitly than it is shown by default with the TreeView control. One possibility is to show line connections between parent and child nodes within the display. Simply set the ShowLines property of the TreeView control to True (by default, this property is set to False):

```
<asp:TreeView ID="TreeView1" Runat="server" Font-Underline="false"
  DataSourceId="XmlDataSource1" ShowCheckBoxes="leaf" ShowLines="True">
  ...
</asp:TreeView>
```

This code gives the result shown in Figure 5-15.

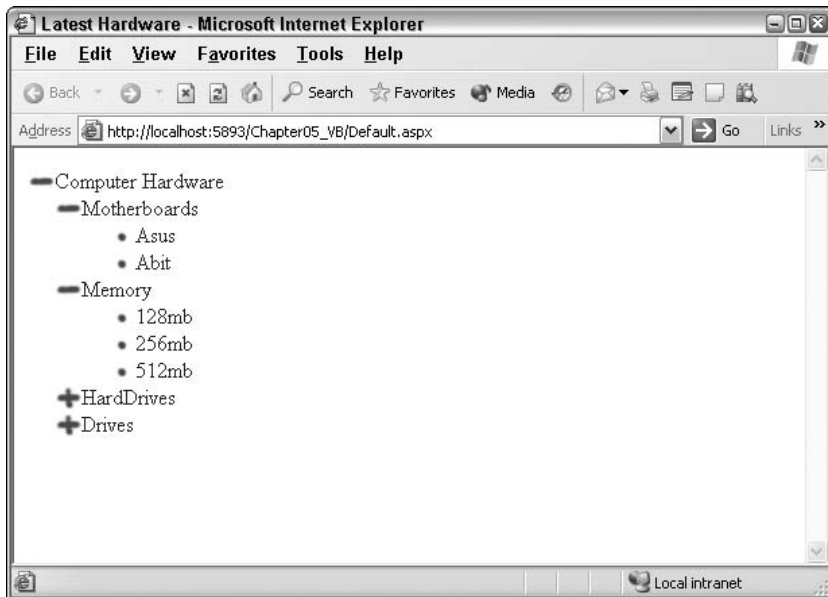


Figure 5-14

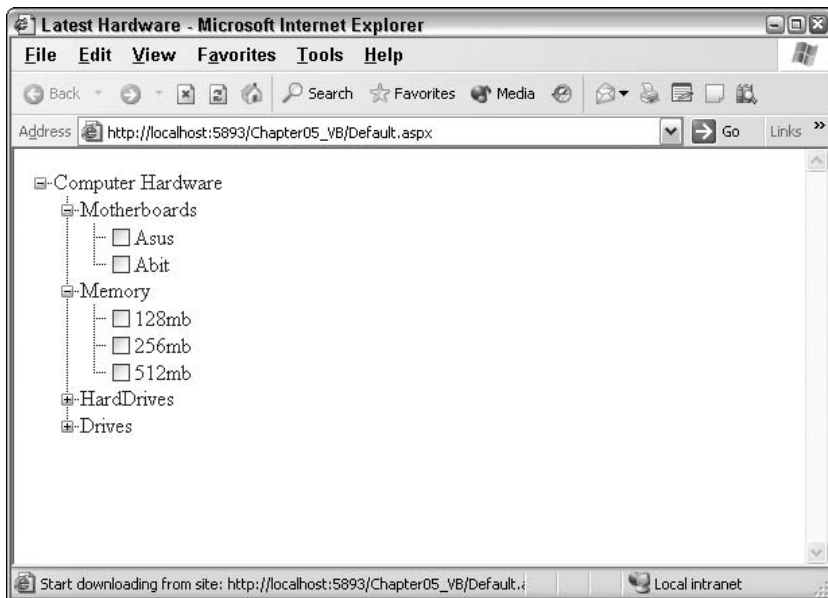


Figure 5-15

If the `ShowLines` property is set to `True`, you can also define your own lines and images within the TreeView control. This is quite easy to do because Visual Studio 2005 provides you with an ASP.NET TreeView Line Image Generator tool. This tool enables you to visually design how you want the lines

and corresponding expanding and collapsing images to appear. After you have it set up as you want, the tool then creates all the necessary files for any of your TreeView controls to use.

To get at the tool, move to the Design view of your file and click open the smart tag for the TreeView control that is on your page. Here you find the option Customize Line Images. Click this and you are presented with the ASP.NET TreeView Line Image Generator dialog (shown in Figure 5-16).

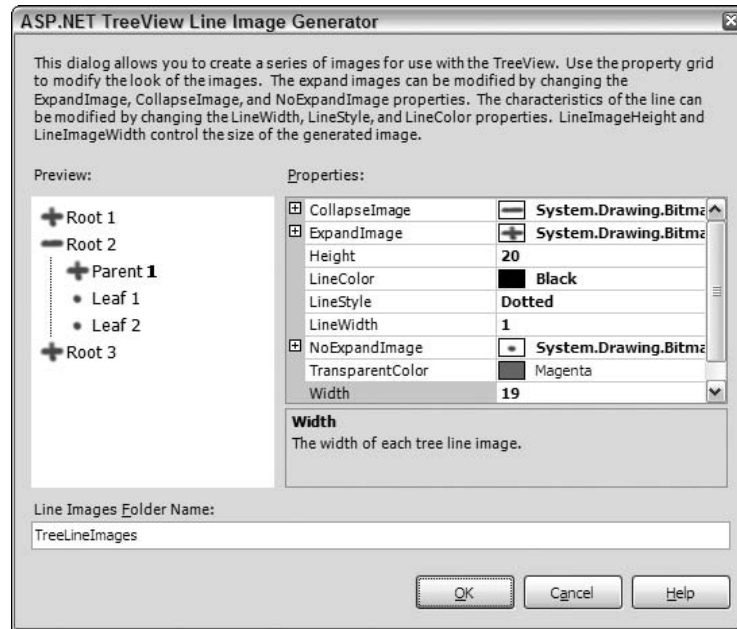


Figure 5-16

From within this dialog, you can select the images used for the nodes that require an Expand, Collapse, or NoCollapse icon. You can also specify the color and style of the lines that connect the nodes. As you create your styles, a sample TreeView control output is displayed for you directly in the dialog based on how your styles are to be applied. The final step is to choose the output of the files that this dialog will create. When you have completed this step, click the OK button. This generates a long list of new files to the folder that you specified in the dialog. By default, the ASP.NET TreeView Line Image Generator wants you to name the output folder `TreeLineImages`, but feel free to name it as you wish. If the folder doesn't exist in the project, you will be prompted to allow Visual Studio to create the folder for you. Once in place, the TreeView control can use your new images and styles by setting the `LineImagesFolderUrl` property as shown here:

```
<asp:TreeView ID="TreeView1" Runat="server" ShowLines="true"
DataSourceId="SiteMapDataSource1" LineImagesFolderUrl="TreeViewLineImages">
```

The important properties are shown in bold. The `ShowLines` property must be set to `True`. After it is set, it uses the default settings displayed earlier, unless you have specified a location where it can retrieve custom images and styles using the `LineImagesFolderUrl` property. As you can see, this simply points to the new folder, `TreeViewLineImages`, that you created and which contains all the new images and styles. Take a look in the folder. It is interesting to see what is output by the tool.

Working with the TreeView control programmatically

So far with the TreeView control, you have learned how to work with the control declaratively. The great thing about ASP.NET is that you are not simply required to work with its components declaratively, but you can also manipulate these controls programmatically.

The TreeView control has an associated `TreeView` class that enables you to completely manage the TreeView control and how it functions from within your code. The next section takes a look at how to use some of the more common ways to control the `TreeView` programmatically.

Expanding and collapsing nodes programmatically

One thing you can do with your TreeView control is to expand or collapse the nodes within the hierarchy programmatically. You can accomplish this by using either the `ExpandAll` or `CollapseAll` methods from the `TreeView` class. Listing 5-12 shows you one of the earlier TreeView controls that you used in Listing 5-6, but with a couple of buttons above it that you now use to initiate the expanding and collapsing of the nodes.

Listing 5-12: Expanding and collapsing the nodes of the TreeView control programmatically

VB

```
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.ExpandAll()
    End Sub

    Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.CollapseAll()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Expand Nodes"
                OnClick="Button1_Click" />
            <asp:Button ID="Button2" Runat="server" Text="Collapse Nodes"
                OnClick="Button2_Click" />
            <br />
            <br />
            <asp:TreeView ID="TreeView1" Runat="server"
                DataSourceId="SiteMapDataSource1">
            </asp:TreeView>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
        </p>
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        TreeView1.ExpandAll();
    }

    void Button2_Click(object sender, System.EventArgs e)
    {
        TreeView1.CollapseAll();
    }
</script>
```

Running this page gives you two buttons above your TreeView control. Clicking the first button invokes the `ExpandAll` method and completely expands the entire list of nodes. Clicking the second button invokes the `CollapseAll` method and completely collapses the list of nodes (see Figure 5-17).

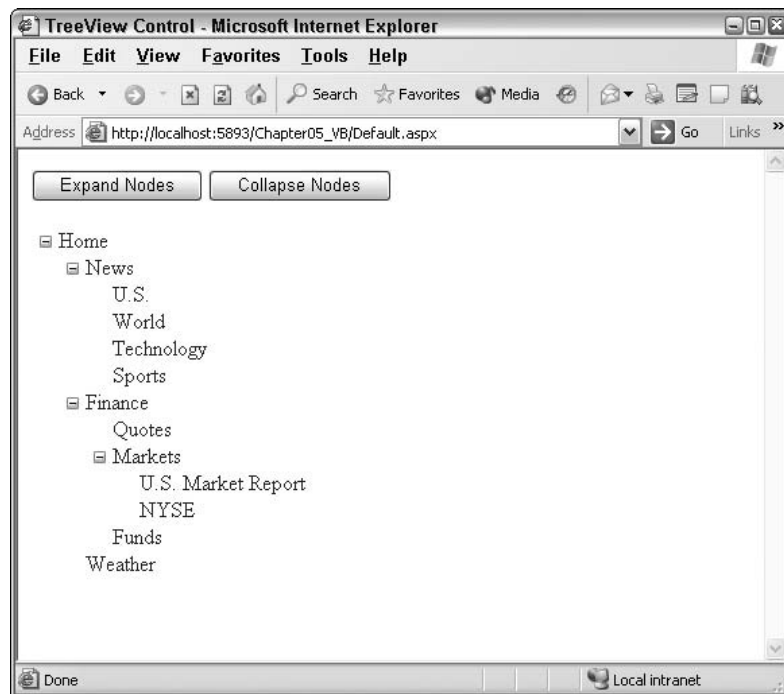


Figure 5-17

The example shown in Listing 5-12 is nice, but it only expands and collapses the nodes on end user actions (when the end user clicks the button). It would be even nicer if you could initiate this action programmatically.

Chapter 5

You might think that you could simply place the `TreeView1.ExpandAll()` command within the `Page_Load` event, but if you try this, you see that it doesn't work. Instead, you use the `OnDataBound` attribute within the `TreeView` control:

```
<asp:TreeView ID="TreeView1" Runat="server"
  DataSourceId="SiteMapDataSource1" OnDataBound="TreeView1_DataBound">
</asp:TreeView>
```

The value of this attribute points to a method in your code, as shown here:

VB

```
Sub TreeView1_DataBound(ByVal sender As Object, ByVal e As System.EventArgs)
    TreeView1.ExpandAll()
End Sub
```

C#

```
void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.ExpandAll();
}
```

Now when you run the page, notice that the `TreeView` control is completely expanded when the page is first loaded in the browser.

You can also expand specific nodes within the tree instead of just expanding the entire list. For this example, use the `TreeView1_DataBound` method you just created. Using the site map from Listing 5-1, change the `TreeView1_DataBound` method so that it appears as shown in Listing 5-13.

Listing 5-13: Expanding specific nodes programmatically

VB

```
Sub TreeView1_DataBound(ByVal sender As Object, ByVal e As System.EventArgs)
    TreeView1.FindNode("Home").Expand()
    TreeView1.FindNode("Home\Finance").Expand()
    TreeView1.FindNode("Home\Finance\Markets").Expand()
End Sub
```

C#

```
void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.FindNode("Home").Expand();
    TreeView1.FindNode("Home\\Finance").Expand();
    TreeView1.FindNode("Home\\Finance\\Markets").Expand();
}
```

In this case, you are using the `FindNode` method and expanding the node that is found. The `FindNode` method takes a `String` value, which is the node and the path of the node that you want to reference. For instance, `TreeView1.FindNode("Home\Finance").Expand()` expands the `Finance` node. To find the node, it is important to specify the entire path from the root node to the node you want to work with (in this case, the `Finance` node). You separate the nodes within the site map path structure with a backslash in between each of the nodes in the site map path (two backslashes if you are working in C#).

Note that you had to expand each of the nodes individually until you got to the `Finance` node. If you simply used `TreeView1.FindNode("Home\Finance\Markets").Expand()` in the `TreeView1_DataBound` method, the `Finance` node would indeed be expanded, but the parent nodes above it (the `Finance` and `Home` nodes) would not have been expanded and you wouldn't see the expanded `Markets` node when invoking the page. (Try it, it's interesting.)

Instead of using the `Expand` method, you could just as easily have used the `Expanded` property and set it to `True`, as shown in Listing 5-14.

Listing 5-14: Expanding nodes programmatically using the `Expanded` property

VB

```
Sub TreeView1_DataBound(ByVal sender As Object, ByVal e As System.EventArgs)
    TreeView1.FindNode("Home").Expanded = True
    TreeView1.FindNode("Home\Finance").Expanded = True
    TreeView1.FindNode("Home\Finance\Markets").Expanded = True
End Sub
```

C#

```
void TreeView1_DataBound(object sender, System.EventArgs e)
{
    TreeView1.FindNode("Home").Expanded = true;
    TreeView1.FindNode("Home\\Finance").Expanded = true;
    TreeView1.FindNode("Home\\Finance\\Markets").Expanded = true;
}
```

Although you focused on the `Expand` method and the `Expanded` property here, you can just as easily work on programmatically collapsing nodes using the `Collapse` method. No `Collapsed` property really exists. Instead, you simply set the `Expanded` property to `False`.

Adding nodes

Another interesting thing you can do with the `TreeView` control is to add nodes to the overall hierarchy programmatically. The `TreeView` control is made up of a collection of `TreeNode` objects. So as you see in previous examples, the `Finance` node is actually a `TreeNode` object that you can work with programmatically. It includes the capability to add other `TreeNode` objects.

A `TreeNode` object typically stores a `Text` and `Value` property. The `Text` property is what is displayed in the `TreeView` control for the end user. The `Value` property is an additional data item that you can use to associate with this particular `TreeNode` object. Another property that you can use (if your `TreeView` control is a list of navigational links) is the `NavigateUrl` property. Listing 5-15 demonstrates how to add nodes programmatically to the same site map from Listing 5-1 that you have been using.

Listing 5-15: Adding nodes programmatically to the `TreeView` control

VB

```
<%@ Page Language="VB" %>
<script runat="server" language="vb">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
```

(continued)

Listing 5-15: *(continued)*

```
        TreeView1.ExpandAll()
    End Sub

    Sub Button2_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        TreeView1.CollapseAll()
    End Sub

    Sub Button3_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myNode As New TreeNode
        myNode.Text = TextBox1.Text
        myNode.NavigateUrl = TextBox2.Text
        TreeView1.FindNode("Home\Finance\Markets").ChildNodes.Add(myNode)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>TreeView Control</title>
</head>
<body>
    <form id="Form1" runat="server">
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Expand Nodes"
                OnClick="Button1_Click" />
            <asp:Button ID="Button2" Runat="server" Text="Collapse Nodes"
                OnClick="Button2_Click" /></p>

        <p>
            <strong>Text of new node:</strong>
            <asp:TextBox ID="TextBox1" runat="server">
            </asp:TextBox>
        </p>

        <p>
            <strong>Desination URL of new node:</strong>
            <asp:TextBox ID="TextBox2" Runat="server">
            </asp:TextBox>
            <br />
            <br />
            <asp:Button ID="Button3" Runat="server" Text="Add New Node"
                OnClick="Button3_Click" />
        </p>

        <p>
            <asp:TreeView ID="TreeView1" runat="server"
                DataSourceId="SiteMapDataSource1">
            </asp:TreeView></p>

        <p>
            <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" /></p>
    </form>
</body>
</html>
```

C#

```

void Button3_Click(object sender, System.EventArgs e)
{
    TreeNode myNode = new TreeNode();
    myNode.Text = TextBox1.Text;
    myNode.NavigateUrl = TextBox2.Text;
    TreeView1.FindNode( "Home\\Finance\\Markets" ).ChildNodes.Add(myNode) ;
}

```

This page contains two text boxes and a new Button control. The first text box is used to populate the `Text` property of the new node that is created. The second text box is used to populate the `NavigateUrl` property of the new node.

If you run the page, you can expand the entire hierarchy by pressing the `Expand Nodes` button. Then you can add additional child nodes to the `Markets` node. To add a new node programmatically, use the `FindNode` method as you did before to find the `Markets` node. When you find it, you can add additional child nodes by using the `ChildNodes.Add` method and pass in a `TreeNode` object instance. Submitting `NASDAQ` in the first text box and `Nasdaq.aspx` in the second text box changes your `TreeView` control, as illustrated in Figure 5-18.

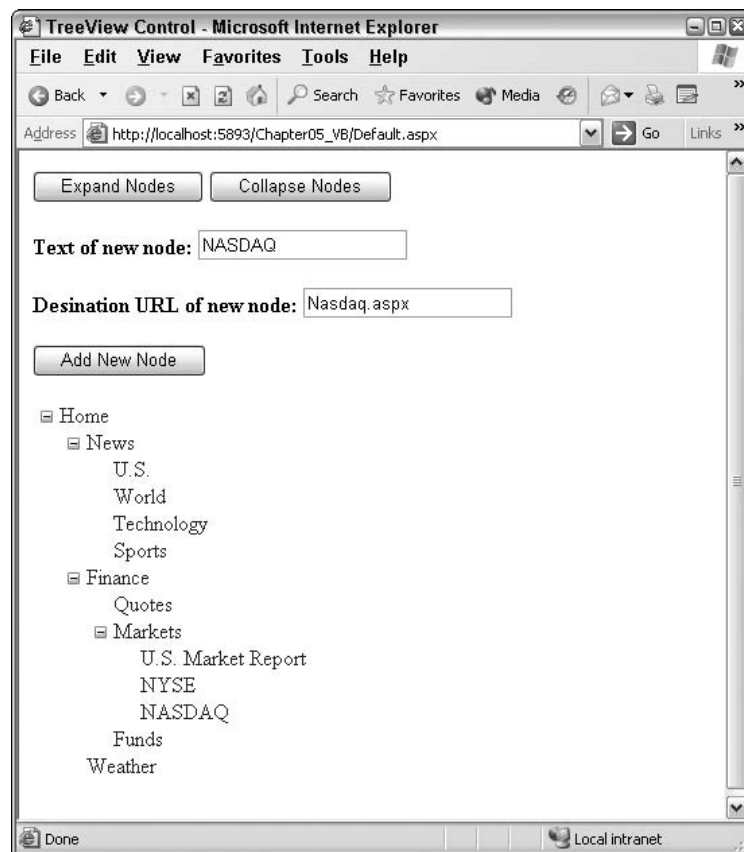


Figure 5-18

After it is added, the node stays added even after the hierarchy tree is collapsed and re-opened. You can also add as many child nodes as you want to the `Markets` node. It is important to note that although you are changing nodes programmatically, this in no way alters the contents of the data source (the XML file, or the `.sitemap` file). These sources remain unchanged throughout the entire process.

Menu Server Control

One of the cooler navigation controls provided with ASP.NET 2.0 is the new Menu server control. This control is ideal for allowing the end user to navigate a larger hierarchy of options while utilizing very little browser real estate in the process. Figure 5-19 shows you what the menu control looks like when it is either completely collapsed or completely extended down one of the branches of the hierarchy.

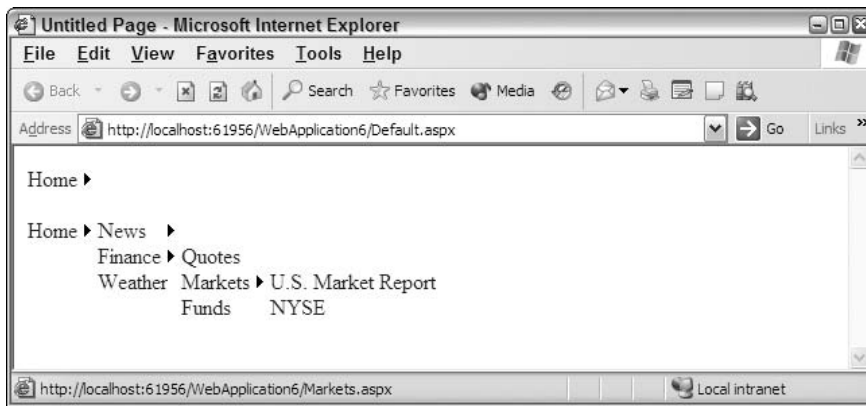


Figure 5-19

From here, you can see that the first Menu control displayed simply shows the Home link with a small arrow to the right of the display. The arrow means that there are more options available that relate to this up-most link in the hierarchy. The second Menu control displayed shows what the default control looks like when the end user works down one of the branches provided by the site map.

The Menu control is an ideal control to use when you have lots of options — whether these options are selections the end user can make or navigation points provided by the application in which they are working. The Menu control can provide a multitude of options and consumes little space in the process.

Using the Menu control in your ASP.NET applications is rather simple. You need to have the control work with a `SiteMapDataSource` control. You can drag and drop the `SiteMapDataSource` control and the Menu control onto the Visual Studio 2005 design surface and connect the two by using the Menu control's `DataSourceId` property. Alternatively, you can create and connect them directly in code. Listing 5-16 shows an example of the Menu control in its simplest form.

Listing 5-16: A simple use of the Menu control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="Sitemapdatasource1" Runat="server" />
        <asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1">
        </asp:Menu>
    </form>
</body>
</html>
```

From this example, you can see that I'm using a SiteMapDataSource control that automatically works with the application's `web.sitemap` file. The only other item included is the Menu control, which uses the typical ID and Runat attributes and the DataSourceID attribute to connect this control with what is retrieved from the SiteMapDataSource control.

Although the default Menu control is pretty simple, you can highly customize how this control looks and works by redefining the properties of the control. The following sections take a look at some examples of how you can modify the appearance and change the behavior of this control.

Applying different styles to the Menu control

By default, the Menu control is pretty plain. If you want to maintain this appearance, you can use what is provided or simply change the font sizes and styles to make it fit in more with your site. There are actually quite a number of ways in which you can modify this control so that it appears unique and fits in with the rest of your site. You can either customize this control's appearance yourself or you can use one of the predefined styles that come with the control.

Using a pre-defined style

Visual Studio 2005 includes some pre-defined styles that you can use with the Menu control to quickly apply a look and feel to the displayed menu of items. Some of the provided styles include `Classic` and `Elegant`. To apply one of these pre-defined styles, you work with the Menu control from the Design view of your page. Within the Design view, highlight the Menu control and expand the control's smart tag. From here, you see a list of options for working with this control. To change the look and feel of the control, click the Auto Format link and select one of the styles.

Performing this operation changes the code of your control by applying a set of style properties. For example, if you select the `Classic` option, you get the results shown in Listing 5-17.

Listing 5-17: Code changes when a style is applied to the Menu control

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1"
  BackColor="#F7F7DE" BorderWidth="1px" BorderColor="#CCCC99" Font-Names="Verdana"
  Font-Size="10pt" BorderStyle="Solid">
  <StaticMenuStyle BorderColor="#CCCC99" Font-Names="Verdana" Font-Size="10pt"
    BackColor="#F7F7DE" BorderStyle="Solid" BorderWidth="1px">
  </StaticMenuStyle>
  <StaticMenuItemStyle BorderColor="#CCCC99" Font-Names="Verdana" Font-Size="10pt"
    BackColor="#F7F7DE" BorderStyle="Solid" BorderWidth="1px">
  </StaticMenuItemStyle>
  <DynamicMenuStyle BorderColor="#CCCC99" Font-Names="Verdana" Font-Size="10pt"
    BackColor="#F7F7DE" BorderStyle="Solid" BorderWidth="1px">
  </DynamicMenuStyle>
  <DynamicMenuItemStyle BorderColor="#CCCC99" Font-Names="Verdana"
    Font-Size="10pt" BackColor="#F7F7DE" BorderStyle="Solid" BorderWidth="1px">
  </DynamicMenuItemStyle>
</asp:Menu>
```

You can see that it added a lot of styles that change the menu items that appear in the control. Figure 5-20 shows how this style selection appears in the browser.

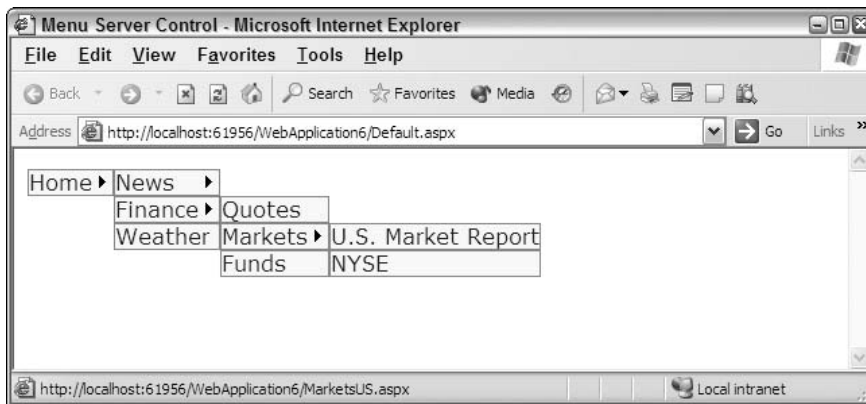


Figure 5-20

Changing the style for static items

The Menu control considers items in the hierarchy to be either *static* or *dynamic*. Static items from this example would be the `Home` link that appears when the page is generated. Dynamic links are the items that appear dynamically when the user hovers the mouse over the `Home` link in the menu. It is possible to change the styles for both these types of nodes in the menu.

To apply a specific style to the static links that appear, you must add a static style element to the Menu control. The Menu control includes the following static style elements:

- ☐ `<StaticHoverStyle>`
- ☐ `<StaticMenuItemStyle>`

- ☐ <StaticMenuStyle>
- ☐ <StaticSelectedStyle>
- ☐ <StaticTemplate>

The important options from this list include the <StaticHoverStyle> and the <StaticMenuItemStyle> elements. The <StaticHoverStyle> is what you use to define the style of the static item in the menu when the end user hovers the mouse over the option. The <StaticMenuItemStyle> is what you use for the style of the static item whether or not the end user is hovering the mouse over the option.

Adding a style to the static item in the menu for when the end user hovers the mouse over the option is illustrated in Listing 5-18.

Listing 5-18: Adding a hover style to static items in the menu control

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1">
  <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"
    BorderWidth="1"></StaticHoverStyle>
</asp:Menu>
```

This little example adds a background color and border to the static items in the menu when the end user hovers the mouse over the item. The result is shown in Figure 5-21.



Figure 5-21

Adding styles to dynamic items

Adding styles to the dynamic items of the menu control is just as easy as it was in adding them to static items. The Menu control includes a number of different elements for modifying the appearance of dynamic items, including the following:

- ☐ <DynamicHoverStyle>
- ☐ <DynamicMenuItemStyle>
- ☐ <DynamicMenuStyle>

- ❑ <DynamicSelectedStyle>
- ❑ <DynamicTemplate>

These elements change menu items the same way as the static versions of these elements, but they change only the items that dynamically pop-out from the static items. Listing 5-19 shows an example of applying the same style to the dynamic items as was applied to the static items.

Listing 5-19: Adding a hover style to dynamic items in the menu control

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1">  
  <StaticHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"  
    BorderWidth="1"></StaticHoverStyle>  
  <DynamicHoverStyle BackColor="DarkGray" BorderColor="Black" BorderStyle="Solid"  
    BorderWidth="1"></DynamicHoverStyle>  
</asp:Menu>
```

This code produces the results shown in Figure 5-22.

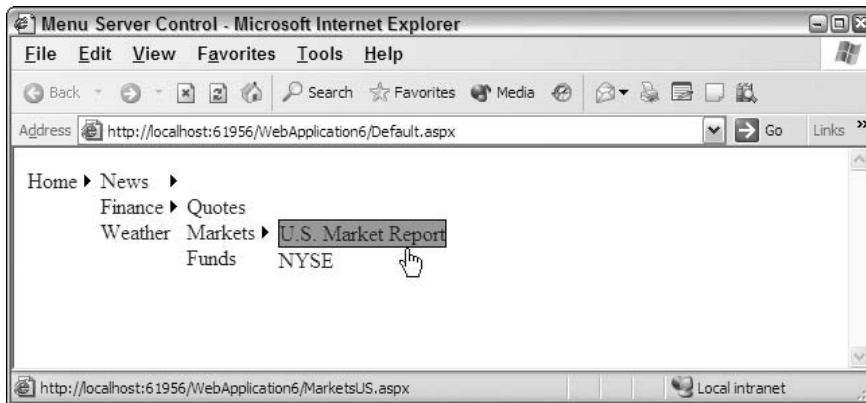


Figure 5-22

Changing the layout of the menu items

By default, the dynamic menu items are displayed from left-to-right. This means that as the items in the menu expand, they are just continually displayed in a vertical fashion. You can actually control this behavior, as there is another option available to you.

The other option is to have the first level of menu items appear directly below the first static item (horizontally). You change this behavior by using the *Orientation* attribute of the Menu control, as shown in Listing 5-20.

Listing 5-20: Forcing the menu items to use a horizontal orientation

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1"  
  Orientation="Horizontal">  
</asp:Menu>
```

This code produces the results shown in Figure 5-23.



Figure 5-23

The Orientation attribute can take a value of Horizontal or Vertical only. The default value is Vertical.

Changing the pop-out symbol

As the default, an arrow is used as the pop-out symbol for the menu items generated, whether they are static or dynamic menu items. This is again shown in Figure 5-24.



Figure 5-24

You are not forced to use this arrow symbol; in fact, you can change it to an image with relatively little work. Listing 5-21 shows how to accomplish this task.

Listing 5-21: Using custom images

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1"
  Orientation="Horizontal" DynamicPopOutImageUrl="myArrow.gif"
  StaticPopOutImageUrl="myArrow.gif">
</asp:Menu>
```

To change the pop-out symbol to an image of your choice, you use the DynamicPopOutImageUrl or StaticPopOutImageUrl properties. The String value these attributes take is simply the path of the image you want to use. Depending on the image used, it produces something similar to what you see in Figure 5-25.

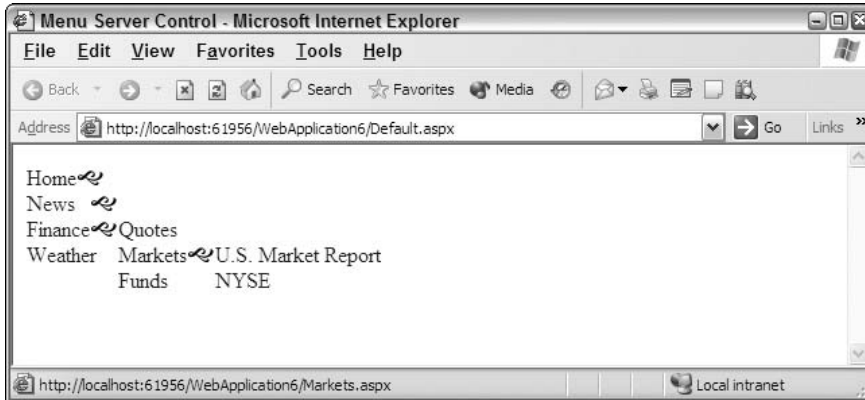


Figure 5-25

Separating menu items with images

Another nice styling option of the Menu control is the capability to add a divider image to the menu items. You use the `StaticBottomSeparatorImageUrl`, `StaticTopSeparatorImageUrl`, `DynamicBottomSeparatorImageUrl`, and `DynamicTopSeparatorImageUrl` properties depending on where you want to place the separator image.

For example, if you wanted to place a divider image under only the dynamic menu items, you would use the `DynamicBottomSeparatorImageUrl` property, as shown in Listing 5-22.

Listing 5-22: Applying divider images to dynamic items

```
<asp:Menu ID="Menu1" Runat="server" DataSourceID="Sitemapdatasource1"
  DynamicBottomSeparatorImageUrl="myDivider.gif">
</asp:Menu>
```

All the properties of the Menu control that define the image to use for the dividers take a `String` value that points to the location of the image. The result of Listing 5-22 is shown in Figure 5-26.

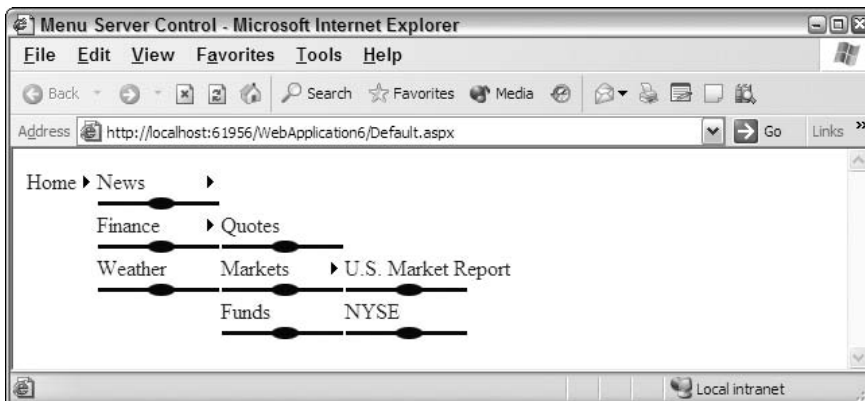


Figure 5-26

Menu Events

The Menu control exposes events such as

- ☐ DataBinding
- ☐ DataBound
- ☐ Disposed
- ☐ Init
- ☐ Load
- ☐ MenuItemClick
- ☐ MenuItemDataBound
- ☐ PreRender
- ☐ Unload

One nice event to be aware of is the `MenuItemClick` event. This event, shown in Listing 5-23, allows you to take some action when the end user clicks one of the available menu items.

Listing 5-23: Using the `MenuItemClick` event

VB

```
Sub Menu1_MenuItemClick(ByVal sender As Object, _  
    ByVal e As System.Web.UI.WebControls.MenuEventArgs)  
  
    ' Code for event here  
  
End Sub
```

C#

```
void Menu1_MenuItemClick(object sender, MenuEventArgs e)  
{  
  
    // Code for event here  
  
}
```

This event uses the `MenuEventArgs` event delegate and provides you access to the text and value of the item selected from the menu.

Binding the Menu control to an XML file

Just as with the `TreeView` control, it is possible to bind the `Menu` control to items that come from other data source controls provided with ASP.NET 2.0. Although most developers are likely to use the `Menu` control to enable end users to navigate to URL destinations, you can also use the `Menu` control to enable users to make selections.

As an example, take the previous XML file, `Hardware.xml`, which was used with the `TreeView` control from Listing 5-8 earlier in the chapter. For this example, the `Menu` control needs to work with an

XmlDataSource control. When the end user makes a selection from the menu, you populate a Listbox on the page with the items selected. The code for this is shown in Listing 5-24.

Listing 5-24: Using the Menu control with an XML file

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Menu1_MenuItemClick(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.MenuEventArgs)

        Listbox1.Items.Add(e.Item.Parent.Value & " : " & e.Item.Value)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Menu Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Menu ID="Menu1" Runat="server" DataSourceID="XmlDataSource1"
            OnMenuItemClick="Menu1_MenuItemClick">
            <Bindings>
                <asp:MenuItemBinding DataMember="Item"
                    TextField="Category"></asp:MenuItemBinding>
                <asp:MenuItemBinding DataMember="Option"
                    TextField="Choice"></asp:MenuItemBinding>
            </Bindings>
        </asp:Menu>
        <p>
            <asp:ListBox ID="Listbox1" Runat="server">
            </asp:ListBox></p>
        <asp:xmldatasource ID="XmlDataSource1" runat="server"
            datafile="Hardware.xml" />
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Menu1_MenuItemClick(object sender, MenuEventArgs e)
    {
        Listbox1.Items.Add(e.Item.Parent.Value + " : " + e.Item.Value);
    }
</script>
```

From this example, you can see that instead of using the `<asp:TreeNodeBinding>` elements, as I did with the TreeView control, the Menu control uses the `<asp:MenuItemBinding>` elements to make connections to items listed in the XML file: `Hardware.xml`. In addition, the root element of the Menu control, the `<asp:Menu>` element, now includes the `OnMenuItemClick` attribute, which points to the event delegate `Menu1_MenuItemClick`.

The `Menu1_MenuItemClick` event includes the event delegate `MenuEventArgs`, which allows you to get at both the values of the child and parent elements selected. For this example, both are used and then populated into the Listbox control, as illustrated in Figure 5-27.



Figure 5-27

SiteMap Data Provider

A whole new series of data providers in the form of `DataSource` controls have been added to ASP.NET 2.0. One of these new `DataSource` controls now at your disposal, which we looked at earlier in the chapter, is the `SiteMapDataSource` control. This new `DataSource` control was developed to work with site maps and the controls that can bind to them.

Some controls don't need a `SiteMapDataSource` control in order to bind to the application's site map (which is typically stored in the `web.sitemap` file). Earlier in the chapter, you saw this in action when using the `SiteMapPath` control. This control was able to work with the `web.sitemap` file directly without the need for this new data provider.

Certain navigation controls, however, such as the `TreeView` control and the `DropDownList` control, require an intermediary `SiteMapDataSource` control to retrieve the site navigation information.

The `SiteMapDataSource` control is simple to use as demonstrated throughout this chapter. The `SiteMapDataSource` control in its simplest form is illustrated here:

```
<asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server" />
```

In this form, the `SiteMapDataSource` control simply grabs the info as a tree hierarchy (as consistently demonstrated so far). Be aware that a number of properties do change how the data is displayed in any control that binds to the data output.

SiteMapViewType

The `SiteMapViewType` property can take one of three available values: `Tree`, `Flat`, or `Path`. Changing this property's value dramatically changes how the view of the site map data is represented in the control that binds to it. The following table describes each of these values.

Value	Description
Tree	This is the default setting for the <code>SiteMapViewType</code> property. The value of <code>Tree</code> assigns the hierarchical site map structure as it is presented in the site map data file.
Flat	This value flattens the hierarchical structure of the data that is presented in the site map data file so that no hierarchical structure is represented. This is ideal setting for controls such as the <code>DropDownList</code> control.
Path	This value presents the navigation structure that shows the data as a hierarchical structure from the current node to the root node (this is similar to how it is presented in the <code>SiteMapPath</code> control).

StartingNodeType

The `StartingNodeType` property sets the depth where the `SiteMapDataSource` control starts retrieving node objects. It is based on the current node. For example, if you were looking at a page in the hierarchy that contained child nodes and you set the `StartingNodeType` to `Root` (which is the default), all the links for the entire hierarchy are displayed. However, if you set the `StartingNodeType` to `Parent`, only the parent node to the node being displayed and all the successive child nodes are displayed. Finally, if you set the `StartingNodeType` to `Current`, only the current node and any child nodes are shown in the hierarchy. This is demonstrated in Listing 5-25. For this example, use the site map from Listing 5-1 and create a page called `Markets.aspx`.

Listing 5-25: Changing the `StartingNodeType` values in `Markets.aspx`

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:SiteMapDataSource ID="SiteMapDataSource1" Runat="server"
            SiteMapViewType="flat" StartingNodeType="root" />
        <asp:BulletedList ID="BulletedList1" Runat="server"
            DisplayMode="HyperLink" DataSourceId="SiteMapDataSource1"
            DataTextField="Title" DataValueField="Url" BulletStyle="Circle" >
        </asp:BulletedList>
    </form>
</body>
</html>
```

For this page, the `StartingNodeType` is set to `Root` for the bulleted list of site links. This gives you the results illustrated in Figure 5-28.

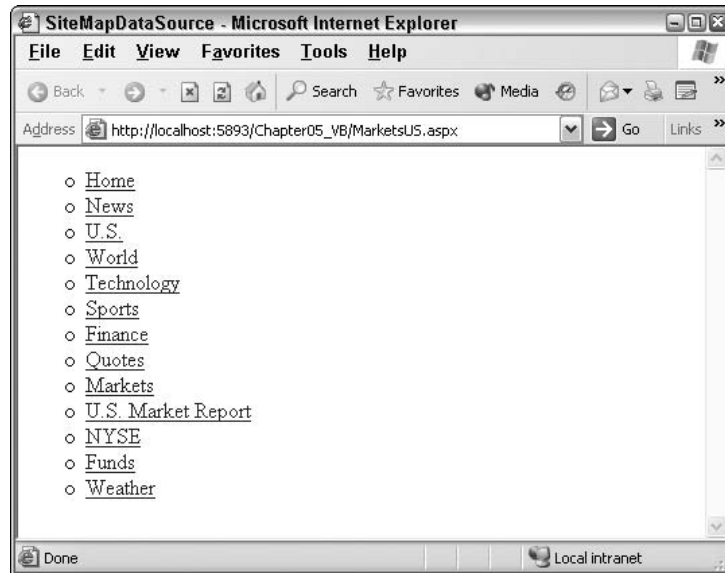


Figure 5-28

Now simply change the `StartingNodeType` to `Parent`, and you get the results illustrated in Figure 5-29.

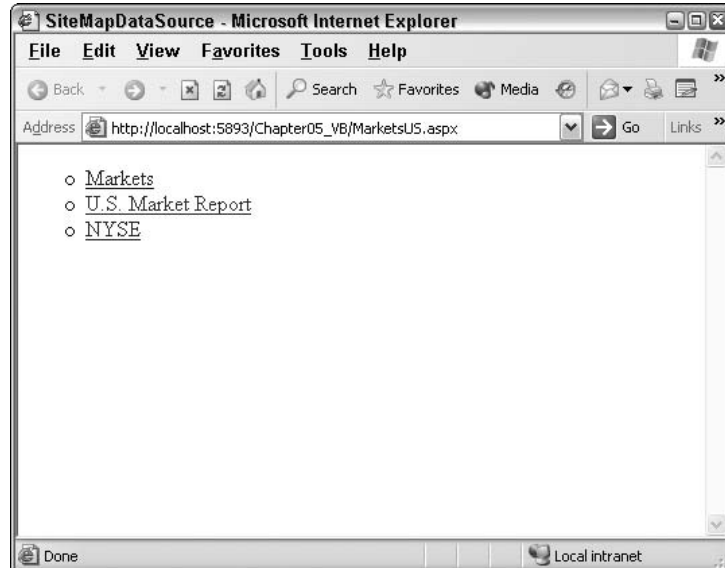


Figure 5-29

And finally, set the `StartingNodeType` to `Current`, and you get the results shown in Figure 5-30.

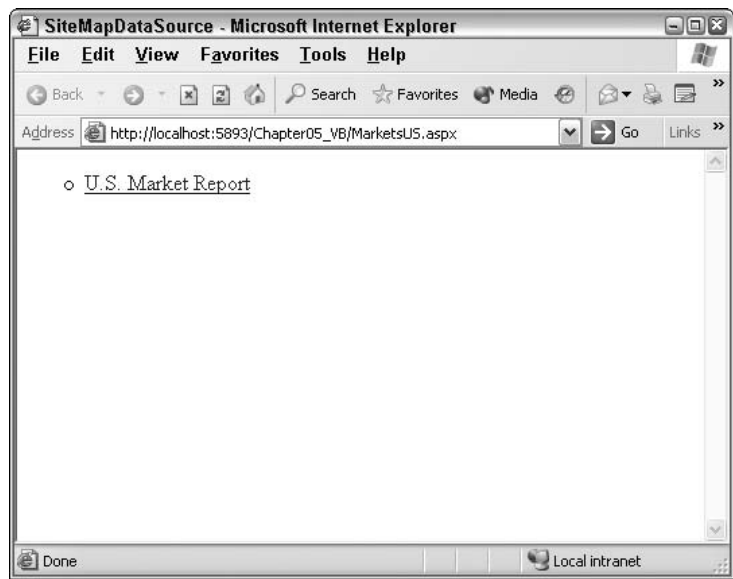


Figure 5-30

SiteMap API

The `SiteMap` class is an in-memory representation of the site’s navigation structure. This is a great class for programmatically working around the hierarchical structure of your site. The `SiteMap` class comes with a couple of objects that make working with the navigation structure easy. These objects are described in the following table.

Object	Description
<code>CurrentNode</code>	Retrieves a <code>SiteMapNode</code> object for the current page.
<code>RootNode</code>	Retrieves a <code>SiteMapNode</code> object that starts from the root node and the rest of the site’s navigation structure.
<code>Provider</code>	Retrieves the default <code>ISiteMapProvider</code> for the current site map.
<code>Providers</code>	Retrieves a collection of available, named <code>ISiteMapProvider</code> objects.

For an example of how to work with some of these `SiteMap` objects, see Listing 5-26, which gives a demonstration of using the `CurrentNode` object.

Listing 5-26: Working with the `CurrentNode` object

```
VB
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
```

```

        Labell1.Text = SiteMap.CurrentNode.Description & "<br>" & _
        SiteMap.CurrentNode.HasChildNodes & "<br>" & _
        SiteMap.CurrentNode.NextSibling.ToString() & "<br>" & _
        SiteMap.CurrentNode.ParentNode.ToString() & "<br>" & _
        SiteMap.CurrentNode.PreviousSibling.ToString() & "<br>" & _
        SiteMap.CurrentNode.RootNode.ToString() & "<br>" & _
        SiteMap.CurrentNode.Title & "<br>" & _
        SiteMap.CurrentNode.Url

    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Labell1" Runat="server"></asp:Label>
    </form>
</body>
</html>

```

C#

```

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

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        Labell1.Text = SiteMap.CurrentNode.Description + "<br>" +
        SiteMap.CurrentNode.HasChildNodes + "<br>" +
        SiteMap.CurrentNode.NextSibling.ToString() + "<br>" +
        SiteMap.CurrentNode.ParentNode.ToString() + "<br>" +
        SiteMap.CurrentNode.PreviousSibling.ToString() + "<br>" +
        SiteMap.CurrentNode.RootNode.ToString() + "<br>" +
        SiteMap.CurrentNode.Title + "<br>" +
        SiteMap.CurrentNode.Url;
    }
</script>

```

As you can see from this little bit of code, by using the `SiteMap` class and the `CurrentNode` object, you can work with a plethora of information regarding the current page. Running this page, you get the following results printed to the screen:

```

The Latest Market Information

True
Funds
Finance
Quotes
Home
Markets
/Chapter05_VB/Markets.aspx

```

Using the `CurrentNode` property, you can actually create your own style of `SiteMapPath` control as illustrated in Listing 5-27.

Listing 5-27: Creating a custom navigation display using the CurrentNode property

VB

```
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Hyperlink1.Text = SiteMap.CurrentNode.ParentNode.ToString()
        Hyperlink1.NavigateUrl = SiteMap.CurrentNode.ParentNode.Url

        Hyperlink2.Text = SiteMap.CurrentNode.PreviousSibling.ToString()
        Hyperlink2.NavigateUrl = SiteMap.CurrentNode.PreviousSibling.Url

        Hyperlink3.Text = SiteMap.CurrentNode.NextSibling.ToString()
        Hyperlink3.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>SiteMapDataSource</title>
</head>
<body>
    <form id="form1" runat="server">
        Move Up:
        <asp:Hyperlink ID="Hyperlink1" Runat="server"></asp:Hyperlink><br />
        <-- <asp:Hyperlink ID="Hyperlink2" Runat="server"></asp:Hyperlink> |
        <asp:Hyperlink ID="Hyperlink3" Runat="server"></asp:Hyperlink> -->
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        Hyperlink1.Text = SiteMap.CurrentNode.ParentNode.ToString();
        Hyperlink1.NavigateUrl = SiteMap.CurrentNode.ParentNode.Url;

        Hyperlink2.Text = SiteMap.CurrentNode.PreviousSibling.ToString();
        Hyperlink2.NavigateUrl = SiteMap.CurrentNode.PreviousSibling.Url;

        Hyperlink3.Text = SiteMap.CurrentNode.NextSibling.ToString();
        Hyperlink3.NavigateUrl = SiteMap.CurrentNode.NextSibling.Url;
    }
</script>
```

When run, this page gives you your own custom navigation structure, as shown in Figure 5-31.

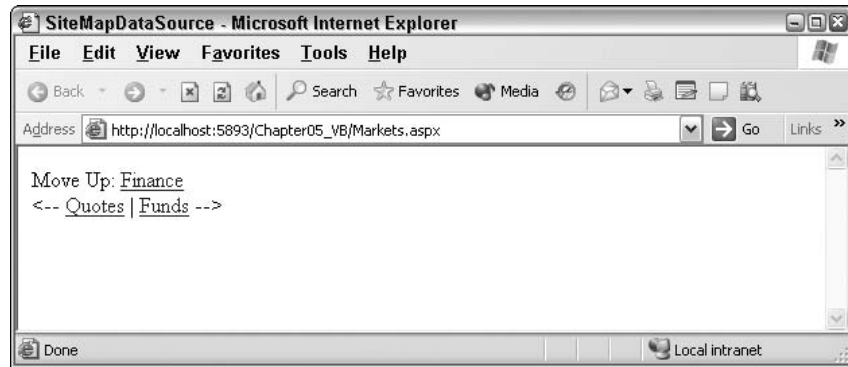


Figure 5-31

Summary

This chapter introduced the new navigation mechanics that ASP.NET 2.0 provides. At the core of the new navigation capabilities is the power to detail the navigation structure in an XML file, which can then be utilized by various navigation controls—such as the new TreeView and SiteMapPath controls.

The powerful functionality that the new navigation capabilities provide saves you a tremendous amount of coding time.

In addition to showing you the core infrastructure for navigation in ASP.NET 2.0, this chapter also described both the new TreeView and SiteMapPath controls and how to use them throughout your applications. The great thing about these new controls is that right out of the box they can richly display your navigation hierarchy and allow the end user to work through the site easily. In addition, these controls are easily changeable so that you can go beyond the standard appearance and functionality that they provide.

6

Working with Master Pages

Visual inheritance is a great new enhancement to your Web pages provided by new additions to ASP.NET 2.0. In effect, you can create a single template page that can be used as a foundation for any number of ASP.NET content pages in your application. These templates, called *master pages*, increase your productivity by making your applications easier to build and to manage after they are built. This chapter takes a close look at how to utilize master pages in your applications to the fullest extent. But first, I explain the advantages of master pages.

Why Do You Need Master Pages?

Most Web sites today have common elements used throughout the entire application or on a majority of the pages within the application. For instance, if you look at the main page of the Reuters News Web site (found at www.reuters.com), you see common elements that are used throughout the entire Web site. These common areas are labeled in Figure 6-1.

screen shot, notice a header section, a navigation section, and a footer section on the page. In fact, pretty much every page within the entire application uses these same elements. Even before master pages, you had ways to put these elements into every page; but in most cases, these other means posed difficulties.

Some developers simply copy and paste the code for these common sections to each and every page that requires them. This works, but it's rather labor intensive. And if you use the copy-and-paste method, whenever a change is required to one of these common sections of the application, you have to go into each and every page to make the change. That's not much fun!

In the Active Server 3.0 days, one popular option was to put all the common sections into what was called an *include file*. You could then place this file within your page like this:

```
<!-- #include virtual="/myIncludes/header.asp" -->
```

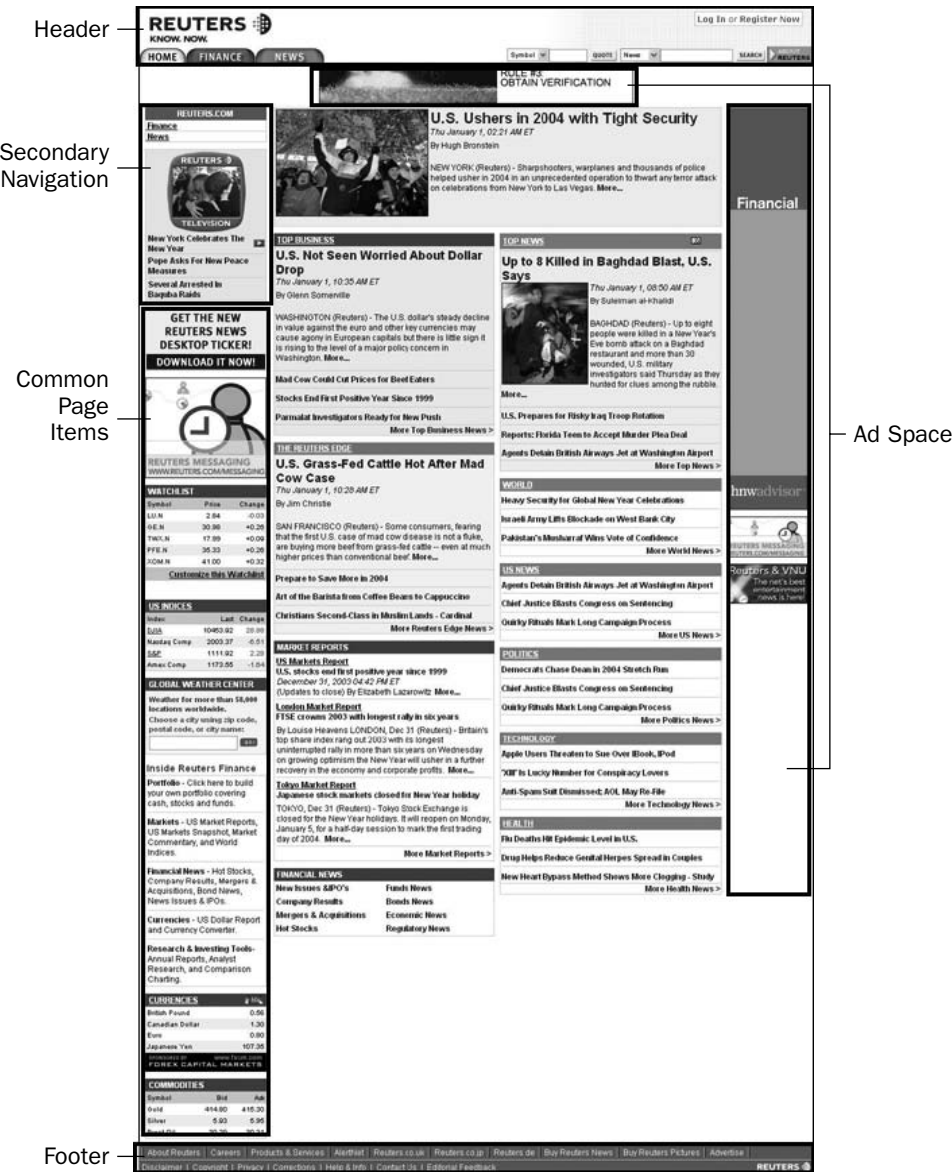


Figure 6-1

In this The problem with using `include` files was that you had to take into account the HTML tags in the header `include` file. These tags had to be closed in the main document or in the footer `include` file. It was usually difficult to keep all the HTML tags in order, especially if multiple people worked on a project. Web pages sometimes displayed strange results because of inappropriate or nonexistent tag closings or openings. It was also difficult to work with `include` files in a visual designer. Using `include` files didn't allow for the developer to see the entire page as it would appear in a browser. The developer ended up developing the page in sections and *hoping* that the pieces would come together as planned.

With the introduction of ASP.NET in 2000, developers started using *user controls* to encapsulate common sections of their Web pages. For instance, you could build a Web page that included header, navigation, and footer sections by simply dragging and dropping these sections of code onto each page that required them. This approach is shown in Figure 6-2.

This technique worked, but it also raised some issues. First, as you can tell from the screen shot, user controls cause problems similar to those related to `include` files. When working in the Design view of your Web page, the common areas of the page display only as gray boxes in Visual Studio .NET. This makes it harder to build a page. You cannot visualize what the page you are building actually looks like until you compile and run the completed page in a browser. Another problem with user controls was that you have to open and close HTML tags properly just as you did with `include` files. I personally like the use of user controls better than `include` files, but they aren't always perfect template pieces for use throughout an application.

In light of these issues, the ASP.NET team has come up with the idea of master pages — an outstanding new way of applying templates to your applications. These pages keep a more distinct line between the common areas that you carry over from page to page and the content areas that are unique on each and every page. Working with master pages is easy and fun. Take a look at some of the basics of master pages in ASP.NET 2.0.

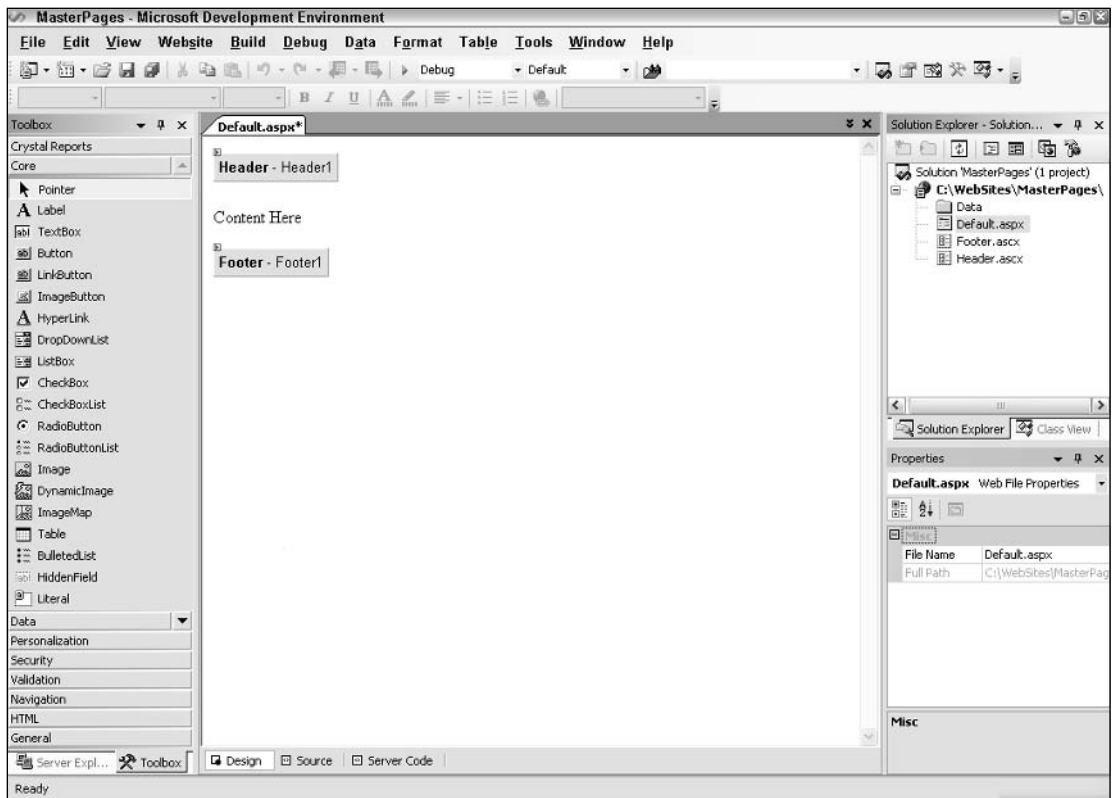


Figure 6-2

The Basics of Master Pages

Master pages are an easy way to provide a template that can be used by any number of ASP.NET pages in your application. In working with master pages, you create a master file that is the template to be used by a *subpage* or *content page*. Master pages use a `.master` file extension and the content pages use the `.aspx` file extension, but are declared content pages within the file's `Page` directive.

Anything that you want to encapsulate within the template should be put into the `.master` file. This can include the header, navigation, and footer sections that are used across the Web application. The content page then contains all the page content except for the master page's elements. At runtime, the ASP.NET engine combines these elements into a single page for the end user. Figure 6-3 shows a diagram of how this process works.

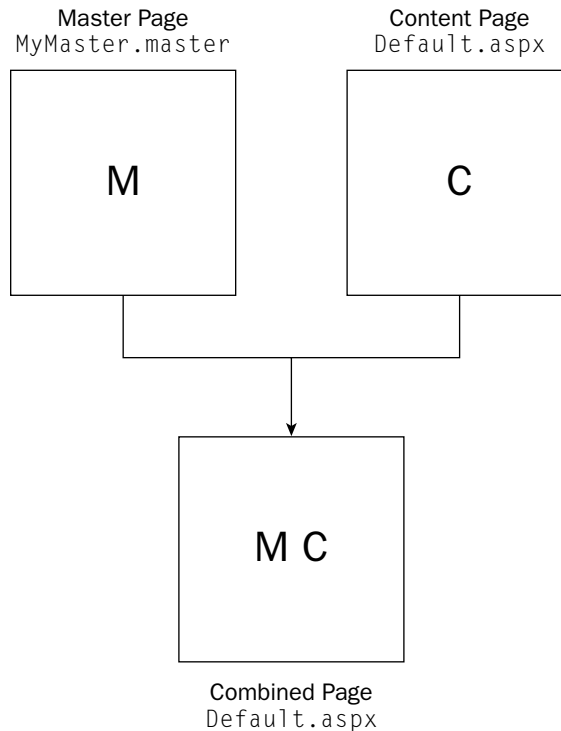


Figure 6-3

One of the nice things about working with master pages is that you can visually see the template in the IDE when you are creating the content pages.

Because you can see the entire page while you are working on it, it is much easier to develop any content pages that use this template. When working in this manner, all the templated items are shown in shaded gray and are not editable while you are working on the content page. The only items that can be altered are clearly shown in the template. These workable areas, called *content areas*, are also defined in the master page itself. Within the master page, you specify the areas of the page that the content pages can use. You can have more than one content area in your master page if you want. Figure 6-4 shows the master page with a couple of content areas shown.

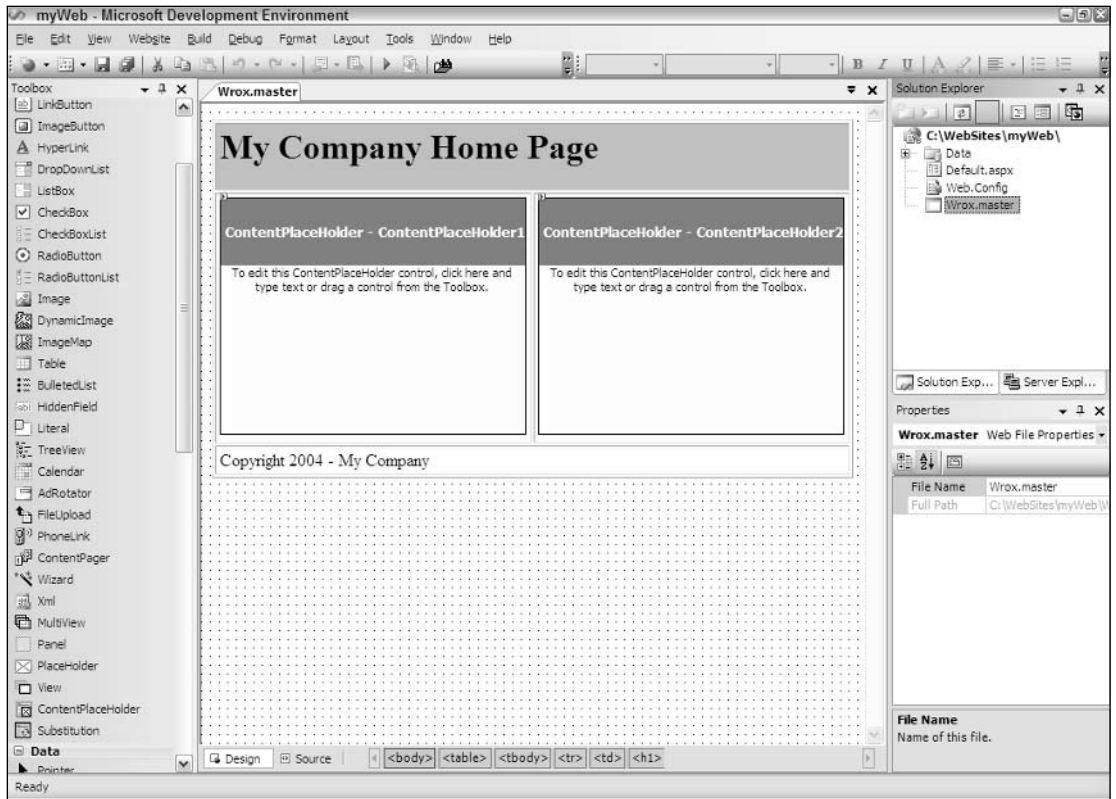


Figure 6-4

With the release of ASP.NET 2.0, master pages are possible because the .NET Framework 2.0 now supports *partial classes*. This is the capability to take two classes and merge them together as a single class at runtime. Using this new capability, the ASP.NET engine is taking two-page classes and bringing them together into a single page at runtime.

Companies and organizations will find using master pages ideal. Many companies have a common look and feel that they apply across their intranet. They can now provide the divisions of their company with a .master file to use when creating a department's section of the intranet. This process makes it quite easy for the company to keep a consistent look and feel across its entire intranet.

Coding a Master Page

You can build the master page from Figure 6-4. You can create one in any text-based editor, such as Notepad, or use the new Visual Studio 2005. In this chapter, I show you how to use Visual Studio 2005.

Master pages work much as regular .aspx pages do, so you can choose the master page option when you add a new file to your application. This is shown in Figure 6-5.

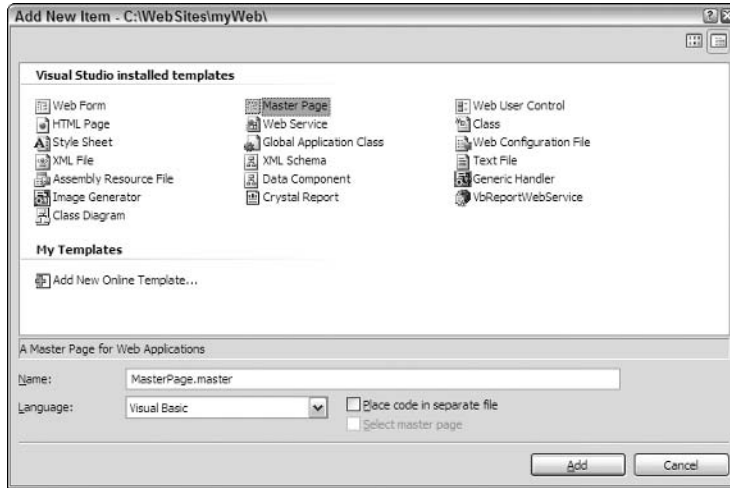


Figure 6-5

Because it's just like any other `.aspx` page, the Add New Item dialog enables you to choose from a master page using the inline coding model or a master page that places its code in a separate file. Not placing your server code in a separate file means that you use the inline code model for the page you are creating. This option creates a single `.master` page. Choosing the option of placing your code in a separate file means that you use the new code-behind model with the page you are creating. The Master Page with Code Separation option creates a single `.master` page along with a `.master.vb` or `.master.cs` file.

A sample master page that uses the inline-coding model is shown in Listing 6-1.

Listing 6-1: A sample master page

```
<%@ Master Language="VB" %>

<script runat="server">

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company Master Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
                    <h1>My Company Home Page</h1>
                </td>
            </tr>
            <tr>
                <td>
```

```

        <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
        Runat="server">
        </asp:ContentPlaceHolder>
    </td>
    <td>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
        Runat="server">
        </asp:ContentPlaceHolder>
    </td>
</tr>
<tr>
    <td colspan="2">
        Copyright 2004 - My Company
    </td>
</tr>
</table>
</form>
</body>
</html>

```

This is a simple master page (it is also lacking any artistic qualities because of author limitations in this area). The great thing about creating master pages in Visual Studio 2005 is that not only can you work with the master page in Code view, you can also switch over to Design view to create your master pages.

Review the code for the master page. The first line is the directive:

```
<%@ Master Language="VB" %>
```

Instead of using the `Page` directive, as you would with a typical `.aspx` page, you use the `Master` directive for a master page. This master page uses only a single attribute, `language`. The `language` attribute's value here is `VB`, but you could also use `C#` (if you are building a C# master page).

You code the rest of the master page just as you would any other `.aspx` page. You can use server controls, raw HTML and text, images, events, or anything else you normally would use for any `.aspx` page. This means that your master page can have a `Page_Load` event as well or any other event that you deem appropriate.

In the code shown in Listing 6-1, notice the use of a new server control — the `<asp:ContentPlaceHolder>` control. This control is used to define the areas of the template where the content page can place its content:

```

<tr>
    <td>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1"
        Runat="server">
        </asp:ContentPlaceHolder>
    </td>
    <td>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
        Runat="server">
        </asp:ContentPlaceHolder>
    </td>
</tr>

```

In the case of this master page, two defined areas exist where the content page can place content. Our master page contains a header and a footer area. It also defines two areas in the page where any inheriting content page can place its own content. Look at how a content page uses this master page.

Coding a Content Page

Now that you have a master page in place in your application, you can use this template for any content pages in your application. To create a new content page within your application, right-click on the application in the Solution Explorer and choose Add New Item.

To create a content page, or a page that will use this master page as its template, you need to select a typical Web Form from the list of options in the Add New Item dialog. Instead of creating a typical Web Form, however, you need to check the Select Master Page check box. This gives you the option later of associating this Web Form to some master page. The Add New Item dialog is shown in Figure 6-6.

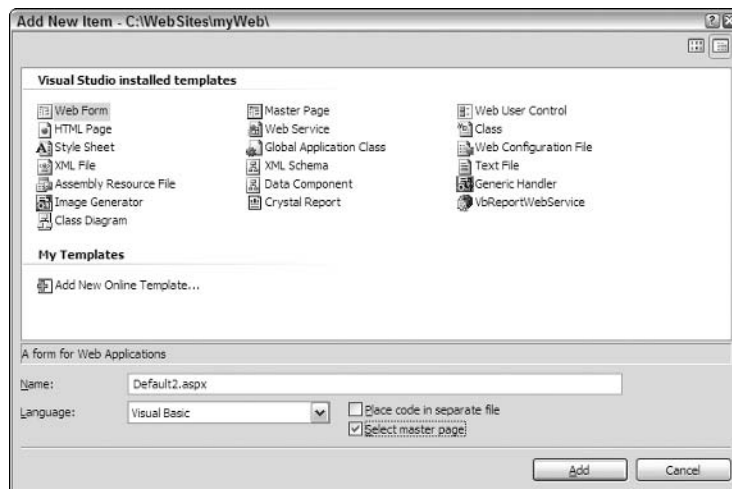


Figure 6-6

After you name your content page and click the Add button in the Add New Item dialog, you are presented with the Select a Master Page dialog, as shown in Figure 6-7.

This dialog allows you to choose the master page from which you want to build your content page. You choose from the available master pages that are contained within your application. Select the new master page that you created in Listing 6-1 and click the OK button. This creates the content page. The created page is a simple .aspx page with only a single line of code contained within the file, as shown in Listing 6-2.

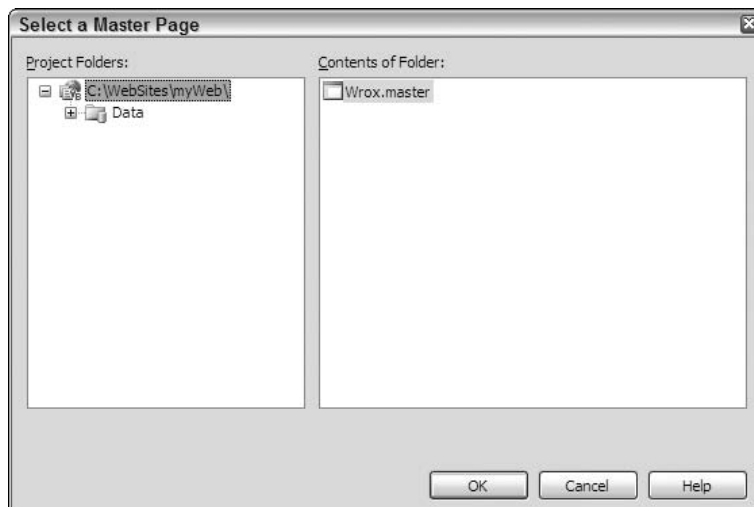


Figure 6-7

Listing 6-2: The created content page

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" Title="Untitled Page" %>
```

This content page is not much different from the typical .aspx page you coded in the past. The big difference is the inclusion of the `MasterPageFile` attribute within the `Page` directive. The use of this attribute specifies that this particular .aspx page inherits from another page. The location of the master page within the application is specified as the value of the `MasterPageFile` attribute.

The other big difference is that it contains neither the `<form id="form1" runat="server">` tag nor any opening or closing HTML tags that would normally be included in a typical .aspx page.

This content page may seem simple, but if you switch to the Design view within Visual Studio 2005, you see the power of using content pages. What you get with visual inheritance is shown in Figure 6-8.

In this screen shot, you can see that just by using the `MasterPageFile` attribute in the `Page` directive, you are able to completely inherit everything that the `Wrox.master` file exposes. All the common areas defined in the master page are shown in gray, whereas the content areas that you specified in the master page using the `<asp:ContentPlaceholder>` server control are shown clearly and available for additional content in the content page. You can add any content to these defined content areas as if you were working with a regular .aspx page. An example of using this .master page for a content page is shown in Listing 6-3.

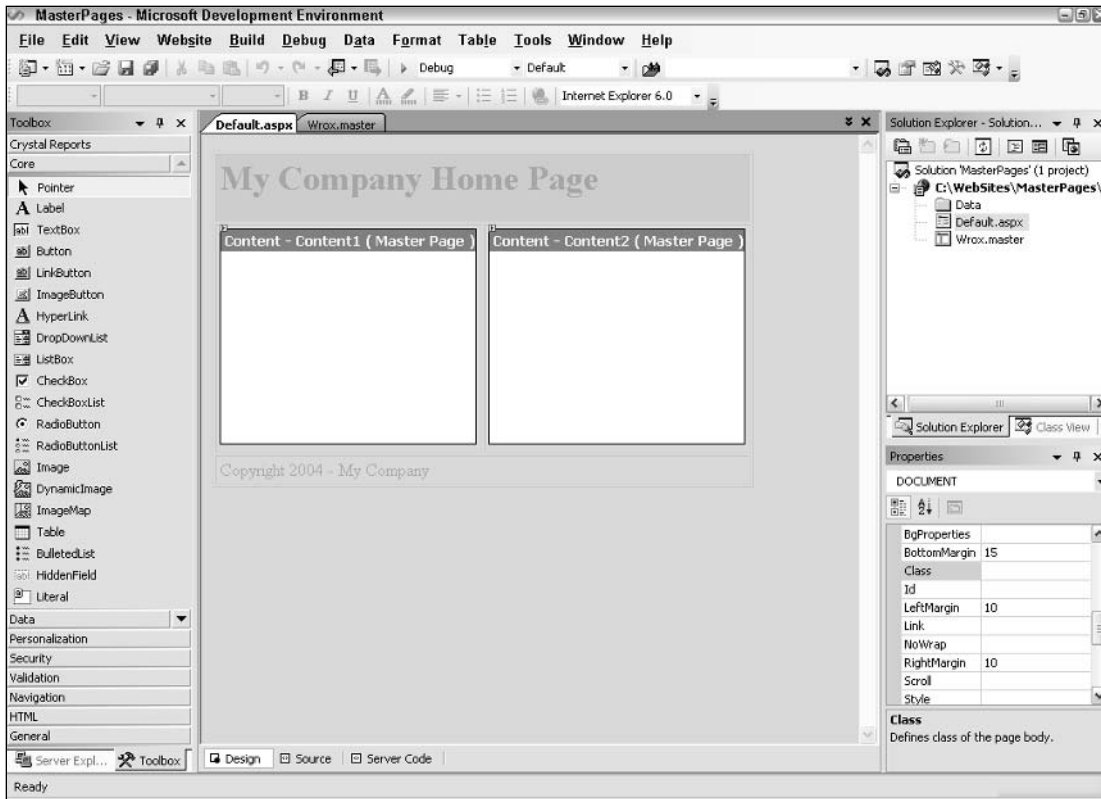


Figure 6-8

Listing 6-3: The content page that uses Wrox.master

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="vb">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "<b>Hello " & TextBox1.Text & "!</b>"
    End Sub
</script>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
    Runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" Runat="server" />
    <br />
    <br />
</asp:Content>
```

```

        <asp:Button ID="Button1" Runat="server" Text="Submit"
            OnClick="Button1_Click" /><br />
        <br />
        <asp:Label ID="Label1" Runat="server" />
    </asp:Content>

    <asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
        Runat="server">
        <asp:Image ID="Image1" Runat="server" ImageUrl="wrox.gif" />
    </asp:Content>

```

C#

```

<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="c#">
    void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "<b>Hello " + TextBox1.Text + "!</b>";
    }
</script>

```

Right away you see some differences. As stated before, this page has no `<form id="form1" runat="server">` tag nor any opening or closing HTML tags. These tags are not included because they are located in the master page. You should also notice a new server control — the `<asp:Content>` server control:

```

    <asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
        Runat="server">
        ...
    </asp:Content>

```

The `<asp:Content>` server control is a defined content area that maps to a specific `<asp:ContentPlaceHolder>` server control on the master page. In this example, you can see that the `<asp:Content>` server control maps itself to the `<asp:ContentPlaceHolder>` server control in the master page that has the ID of `ContentPlaceHolder1`. Within the content page, you don't have to worry about specifying the location of the content because this is completely defined within the master page. Therefore, your only concern is to place the appropriate content within the provided content sections, allowing the master page to do most of the work for you.

Just as when you work with any typical `.aspx` page, you can create any event handlers for your content page. In this case, you are using just a single event handler — the button-click when the end user submits the form. Finally, the created `.aspx` page that includes the master page and content page material is shown in Figure 6-9.

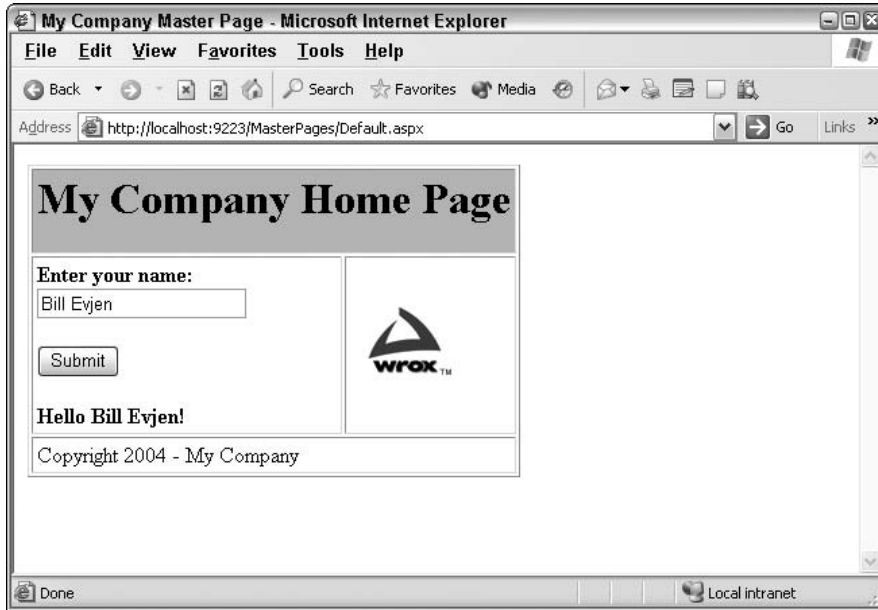


Figure 6-9

Mixing page types and languages

One interesting point: When you use master pages, you are not tying yourself to a specific coding model (inline or code-behind) nor are you tying yourself to the use of a specific language. You can feel free to mix these elements within your application knowing that they all work well.

You could use the master page created earlier, knowing that it was created using the inline-coding model, and then build your content pages using the code-behind model. Listing 6-4 shows a content page created using a Web Form that uses the code-behind option.

Listing 6-4: A content page that uses that code-behind model

.aspx (VB)

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" AutoEventWireup="false"
    CompileWith="MyContentPage.aspx.vb" ClassName="MyContentPage_aspx" %>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
    Runat="server">
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" Runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" Runat="server" Text="Submit"
        OnClick="Button1_Click" /><br />
    <br />
```

```

    <asp:Label ID="Label1" Runat="server" />
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
    Runat="server">
    <asp:Image ID="Image1" Runat="server" ImageUrl="ineta.JPG" />
</asp:Content>

```

VB Code-Behind

```

Imports Microsoft.VisualBasic

Partial Class MyContentPage_aspx

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "<b>Hello " & TextBox1.Text & "!</b>"
    End Sub

End Class

```

C# Code-Behind

```

using System;
using System.Configuration;
using System.Web;
using System.Web.Caching;
using System.Web.SessionState;
using System.Web.Security;
using System.Web.Profile;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class MyContentPage_aspx
{
    void Button1_Click (object sender, System.EventArgs e)
    {
        Label1.Text = "<b>Hello " + TextBox1.Text + "!</b>";
    }
}

```

Even though the master page is using the inline-coding model, you can easily create content pages (such as the page shown in Listing 6-4) that use the code-behind model. The pages will still work perfectly.

Not only can you mix the coding models when using master pages, you can also mix the programming languages. Just because you build a master page in C# doesn't mean that you are required to use C# for all the content pages that use this master page. You can also build content pages in Visual Basic. For a good example, create a master page in C# that uses the `Page_Load` event handler and then create a content page in Visual Basic. Run the page and it works perfectly well. This means that even though you might have a master page in one of the available .NET languages, the programming teams that build applications from the master page can use whatever .NET language they want. You have to love the openness that the .NET Framework offers!

Specifying which master page to use

You just observed that it is pretty easy to specify at page level which master page to use. In the `Page` directive of the content page, you simply use the `MasterPageFile` attribute:

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
```

Besides specifying the master page that you want to use at the page level, you have a second way to specify which master page you want to use in the `web.config` file of the application. This is shown in Listing 6-5.

Listing 6-5: Specifying the master page in the web.config file

```
<configuration>
  <system.web>
    <pages masterPageFile="~/Wrox.master" />
  </system.web>
</configuration>
```

Specifying the master page in the `web.config` file causes every single Web Form you create in the application to inherit from the specified master page. If you declare your master pages in this manner, you can create any number of content pages that use this master page. The content pages' `Page` directive must be constructed in the following manner:

```
<%@ Page Language="VB" %>
```

You can easily override the application-wide master page specification by simply declaring a different master page within your content page:

```
<%@ Page Language="VB" MasterPageFile="~/MyOtherCompany.master" %>
```

By specifying the master page in the `web.config`, you are really saying that you want *all* the `.aspx` pages to use this master page. So if you create a normal Web Form and run it, you get an error stating:

```
Literal content ('<html>') is not allowed on a content page.
```

The application is treating this page as a content page when you really intended it to be a normal `.aspx` page. To get around this, you make the following declaration in the `Page` directive of the page:

```
<%@ Page Language="VB" MasterPageFile="" %>
```

To make this page work without using the default master page, you simply override the master declaration with a master value of nothing. This causes the page to be generated without the use of the master page, and the page then works correctly.

Working with the page title

When you create content pages in your application, by default all the content pages automatically use the title that is declared in the master page. For instance, you have primarily been using a master page

with the title `My Company Master Page`. Every content page that is created using this particular master page also use the same `My Company Master Page` title. You have a way around this, however.

In the code of the content page, you can work with the `Master` object. The `Master` object conveniently has a property called `Title`. The value of this property is the page title that is used for the content page. You code it as shown in Listing 6-6.

Listing 6-6: Coding a custom page title for the content page**VB**

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="vb">
    Sub Page_LoadComplete(ByVal sender As Object, ByVal e As System.EventArgs)
        Master.Page.Title = "This is a title from the content page"
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/wrox.master" %>

<script runat="server">
    void Page_LoadComplete(object sender, EventArgs e)
    {
        Master.Page.Title = "This is a title from the content page";
    }
</script>
```

Working with controls and properties from the master page

When working with master pages from a content page, you actually have good access to the controls and the properties that the master page exposes. The master page, when inherited by the content page, exposes a property called `Master`. You use this property to get at control values or custom properties that are contained in the master page itself.

To see an example of this, create a GUID (unique identifier) in the master page that you can retrieve on the content page that is using the master. For this example, use the master page that was created in Listing 6-1, but with the addition of a `Label` server control and the `Page_Load` event. This is shown in Listing 6-7.

Listing 6-7: A master page that creates a GUID on the first request**VB**

```
<%@ Master Language="VB" %>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Label1.Text = System.Guid.NewGuid().ToString()
        End If
    End Sub
</script>
```

[illegible]

```
void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        Label1.Text = System.Guid.NewGuid().ToString();
    }
}
```

Listing 6-8: Getting at the Label's Text value in the content page**VB**

```

<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>

<script runat="server" language="vb">
    Sub Page_LoadComplete(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = CType(Master.FindControl("Label1"), Label).Text
    End Sub

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label2.Text = "<b>Hello " & TextBox1.Text & " !</b>"
    End Sub
</script>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
    Runat="server">
    <b>Your GUID number from the master page is:<br />
    <asp:Label ID="Label1" Runat="server" /></b><p>
    <b>Enter your name:</b><br />
    <asp:Textbox ID="TextBox1" Runat="server" />
    <br />
    <br />
    <asp:Button ID="Button1" Runat="server" Text="Submit"
        OnClick="Button1_Click" /><br />
    <br />
    <asp:Label ID="Label2" Runat="server" />
</asp:content>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
    Runat="server">
    <asp:Image ID="Image1" Runat="server" ImageUrl="Wrox.gif" />
</asp:Content>

```

C#

```

<%@ Page Language="C#" MasterPageFile="~/wrox.master" %>

<script runat="server">

    void Page_LoadComplete(object sender, EventArgs e)
    {
        Label1.Text = (Master.FindControl("Label1") as Label).Text;
    }

    void Button1_Click(object sender, EventArgs e)
    {
        Label2.Text = "<b>Hello " + TextBox1.Text + " !</b>";
    }
</script>

```

In this example, the master page in Listing 6-7 first creates a GUID that it then stores as a text value in a Label server control on the master page itself. The id of this Label control is `Label1`. The master page

only generates this GUID on the first request for this particular content page. You want to populate one of the content page's controls with this value.

The interesting thing about the content page is that you put code in the `Page_LoadComplete` event handler so that you can get at the GUID value that is on the master page. This new event handler in ASP.NET 2.0 is fired after the `Page_Load` event is fired. I cover event ordering later, but the `Page_Load` event in the content page always fires before the `Page_Load` event in the master page. In order to get at the newly created GUID (if it is created in the master page's `Page_Load` event), you have to get the GUID in an event that comes after this — and that is where the `Page_LoadComplete` comes into play. So within the content page's `Page_LoadComplete` event, you populate a Label server control within the content page itself. Note that the Label control in the content page has the same `id` as the Label control in the master page, but this doesn't make a difference. You can differentiate between them with the use of the `Master` property.

Not only can you get at the server controls that are in the master page in this way, you can get at any custom properties that the master page might expose as well. Look at the master page shown in Listing 6-9; it uses a custom property for the `<h1>` section of the page.

Listing 6-9: A master page that exposes a custom property

VB

```
<%@ Master Language="VB" %>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            Label1.Text = Guid.NewGuid().ToString()
        End If
    End Sub

    Dim m_PageHeadingTitle As String = "My Company"

    Public Property PageHeadingTitle() As String
        Get
            Return m_PageHeadingTitle
        End Get
        Set(ByVal Value As String)
            m_PageHeadingTitle = Value
        End Set
    End Property
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>My Company Master Page</title>
</head>
<body>
    <form id="Form1" runat="server">
        <table cellpadding="3" border="1">
            <tr bgcolor="silver">
                <td colspan="2">
```

[illegible]

C#

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

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            Label1.Text = System.Guid.NewGuid().ToString();
        }
    }
}
```

```
string m_PageHeadingTitle = "My Company";
```

```
public string PageHeadingTitle
{
    get
    {
        return m_PageHeadingTitle;
    }
    set
    {
        m_PageHeadingTitle = value;
    }
}
```

</script>

In this master page example, the master page is exposing the property you created called `PageHeadingTitle()`. You also assign a default value to this property of "My Company". You then place it within the HTML of the master page file between some `<h1>` elements. This makes the default value become the heading used on the page within the master page template. Although the master page already has a value it uses for the heading, any content page that is using this master page can override the `<h1>` title heading. The process is shown in Listing 6-10.

Listing 6-10: A content page that overrides the property from the master page

VB

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server" language="vb">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Master.PageHeadingTitle = "My Company - Division X"
    End Sub
</script>
```

C#

```
<%@ Page Language="C#" MasterPageFile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Master.PageHeadingTitle = "My Company - Division X"
    }
</script>
```

From the content page, you can assign a value to the property that is exposed from the master page by the use of the `Master` property. The result is shown earlier in Figure 6-9. As you can see, this is quite simple to achieve. Remember that not only can you get at any public properties that the master page might expose, but you can also retrieve any methods that the master page contains as well.

Earlier, I showed you how to get at the server controls that are on the master page by using the `FindControl` method. The `FindControl` method works fine, but it is a late-bound approach. Using the mechanics I just illustrated with the use of public properties shown in Listing 6-9, you have another approach to expose any server controls on the master page. You may find this approach to be more effective.

To do this, you simply expose the server control as a public property as shown in Listing 6-11.

Listing 6-11: Exposing a server control from a master page as a public property

VB

```
<%@ Master Language="VB" %>

<script runat="server" language="vb">
    Public Property MasterPageLabel1() As Label
        Get
            Return Label1
        End Get
    End Get
```

```

        Set(ByVal Value As Label)
            Label1 = Value
        End Set
    End Property
</script>

```

C#

```
<%@ Master Language="VB" %>
```

```

<script runat="server" language="C#">
    public Label MasterPageLabel
    {
        get
        {
            return Label1;
        }
        set
        {
            Label1 = value;
        }
    }
</script>

```

In this case, a public property called `MasterPageLabel1` returns an instance of the `Label` control that uses the `id` of `Label1`. You can now create an instance of the `MasterPageLabel1` property on the content page and override any of the attributes of the `Label` server control. So if you want to increase the size of the GUID that the master page creates and displays in the `Label1` server control, you can simply override the `Font.Size` attribute of the `Label` control as shown in Listing 6-12.

Listing 6-12: Overriding an attribute from the Label control that is on the master page**VB**

```

<%@ page language="VB" masterpagefile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

```

```

<script runat="server" language="vb">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Master.MasterPageLabel1.Font.Size = 25
    End Sub
</script>

```

C#

```

<%@ page language="C#" masterpagefile="~/Wrox.master" %>
<%@ MasterType VirtualPath="~/Wrox.master" %>

```

```

<script runat="server" language="C#">
    void Page_Load(object sender, EventArgs e)
    {
        Master.MasterPageLabel1.Font.Size = 25;
    }
</script>

```

This approach may be the most effective way to get at any server controls that the master page exposes to the content pages.

Specifying Default Content in the Master Page

As you have seen, the master page enables you to specify content areas that the content page can use. Master pages can consist of just one content area, or they can be made up of multiple content areas. The nice thing about content areas is that when you create a master page, you can specify default content for the content area. This default content can then be left in place and utilized by the content page if you choose not to override it. Listing 6-13 shows a master page that specifies some default content within a content area.

Listing 6-13: Specifying default content in the master page

```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>My Company</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" Runat="server">
            Here is some default content
        </asp:ContentPlaceHolder><p>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder2" Runat="server">
            Here is some more default content
        </asp:ContentPlaceHolder></p>
    </form>
</body>
</html>
```

To place default content within one of the content areas of the master page, you simply put it in the ContentPlaceHolder server control on the master page itself. Any content page that inherits this master page also inherits the default content. Listing 6-14 shows a content page that overrides just one of the content areas from this master page.

Listing 6-14: Overriding some default content in the content page

```
<%@ Page Language="VB" MasterPageFile="~/MasterPage.master" %>

<asp:Content ID="Content2" ContentPlaceHolderId="ContentPlaceHolder2"
    Runat="server">
    This is new content
</asp:Content>
```

This code creates a page with one content area that shows content coming from the master page itself, in addition to other content that comes from the content page (see Figure 6-10).

The other interesting point when you work with content areas in the Design mode of Visual Studio 2005 is that the smart tag shown for the content area enables you to do two things (shown in Figure 6-11).

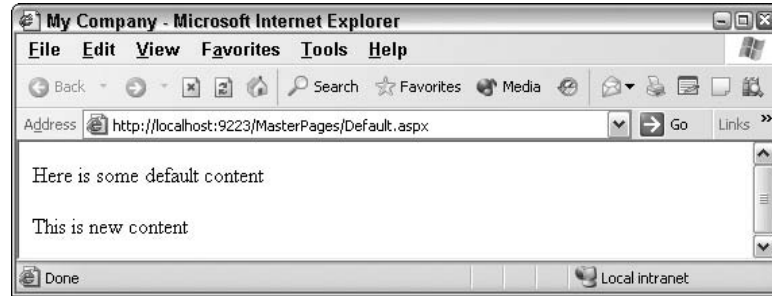


Figure 6-10

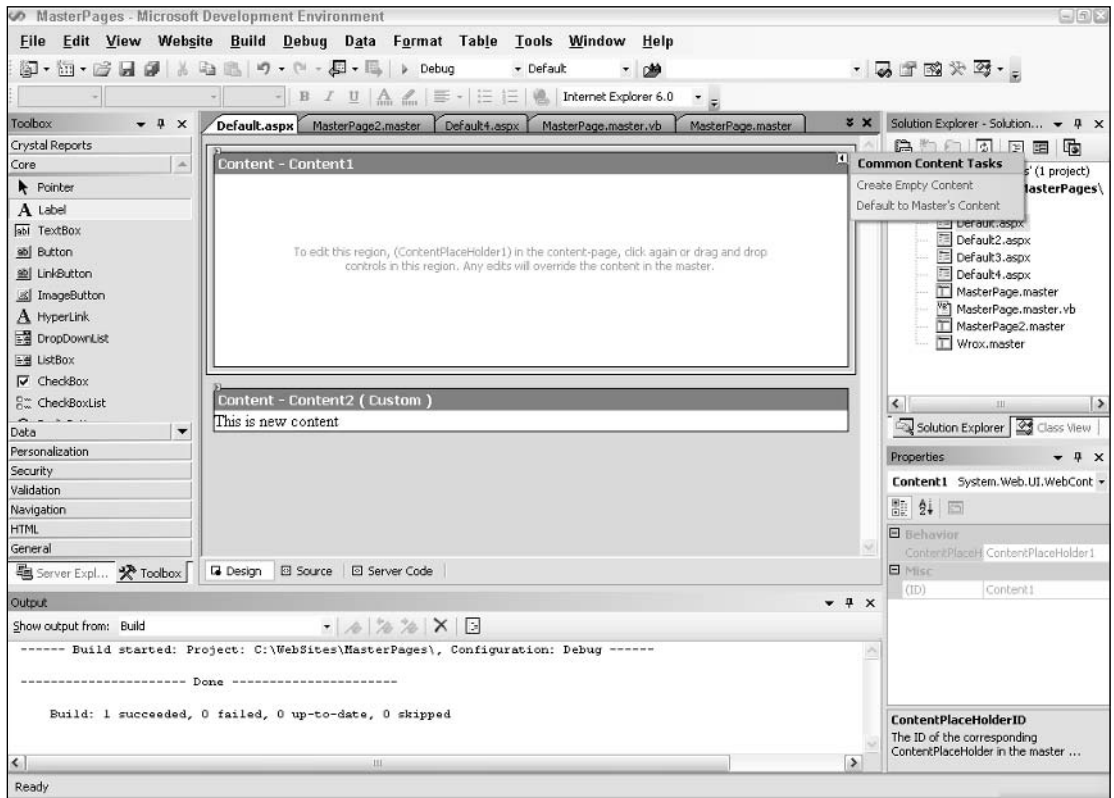


Figure 6-11

The first option is Create Empty Content. This option allows you to override the master page content, but at the same time, keep the content area empty. The second option, Default to Master's Content, enables you to return the default content that the master page exposes to the content area. This option erases whatever content you have already placed in the content area and simply returns the default content.

Nesting Master Pages

I hope you see the power that master pages provide to help you create templated Web applications. So far, you have been creating a single master page that the content page can use. Most companies and organizations, however, are not just two layers. Many divisions and groups exist within the organization that might want to use variations of the master by, in effect, having a master page within a master page. With ASP.NET 2.0 this is quite possible.

For example, imagine that Reuters is creating a master page to be used throughout the entire company intranet. Not only does the global Reuters entity want to implement this master page company-wide, but various divisions within Reuters also want to provide templates for the subsections of the intranet directly under their control. Reuters Europe and Reuters America, for example, each wants its own unique master page, as illustrated in Figure 6-12.

To do this, the creators of the Reuters Europe and Reuters America master page simply create a master page that inherits from the global master page. All the files are shown here starting with Listing 6-15.

Listing 6-15: The main master page

ReutersMain.master

```
<%@ Master Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <p><asp:Label ID="Label1" Runat="server" BackColor="LightGray"
            BorderColor="Black" BorderWidth="1px" BorderStyle="Solid"
            Font-Size="XX-Large">Reuters</asp:Label></p>
        <asp:ContentPlaceHolder ID="ContentPlaceHolder1" Runat="server">
            </asp:ContentPlaceHolder>
        </form>
    </body>
</html>
```

This is a simple master page, but excellent for showing you how this nesting capability works. The main master page is the master page used globally in the company. It has the ContentPlaceHolder server control with the ID of ContentPlaceHolder1.

Listing 6-16 illustrates how you can work with this main master from a submaster file.

Listing 6-16: The submaster page

ReutersEurope.master

```
<%@ Master MasterPageFile="~/ReutersMain.master" %>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder1"
    Runat="server">
    <asp:Label ID="Label1" Runat="server" BackColor="#E0E0E0" BorderColor="Black"
        BorderStyle="Dotted" BorderWidth="2px" Font-Size="Large">
```

```
Reuters Europe</asp:Label><br /><hr />

<asp:ContentPlaceHolder ID="ContentPlaceHolder2" Runat="server">
</asp:ContentPlaceHolder>
</asp:Content>
```

When creating the submaster page, notice that Visual Studio 2005 isn't as friendly when it creates this file for you. This is because Visual Studio 2005 is not expecting the creation of a submaster page. Therefore, to create your submaster page, first create a normal master page and remove all the content in the file except for the directive line. Then you create a Content server control.

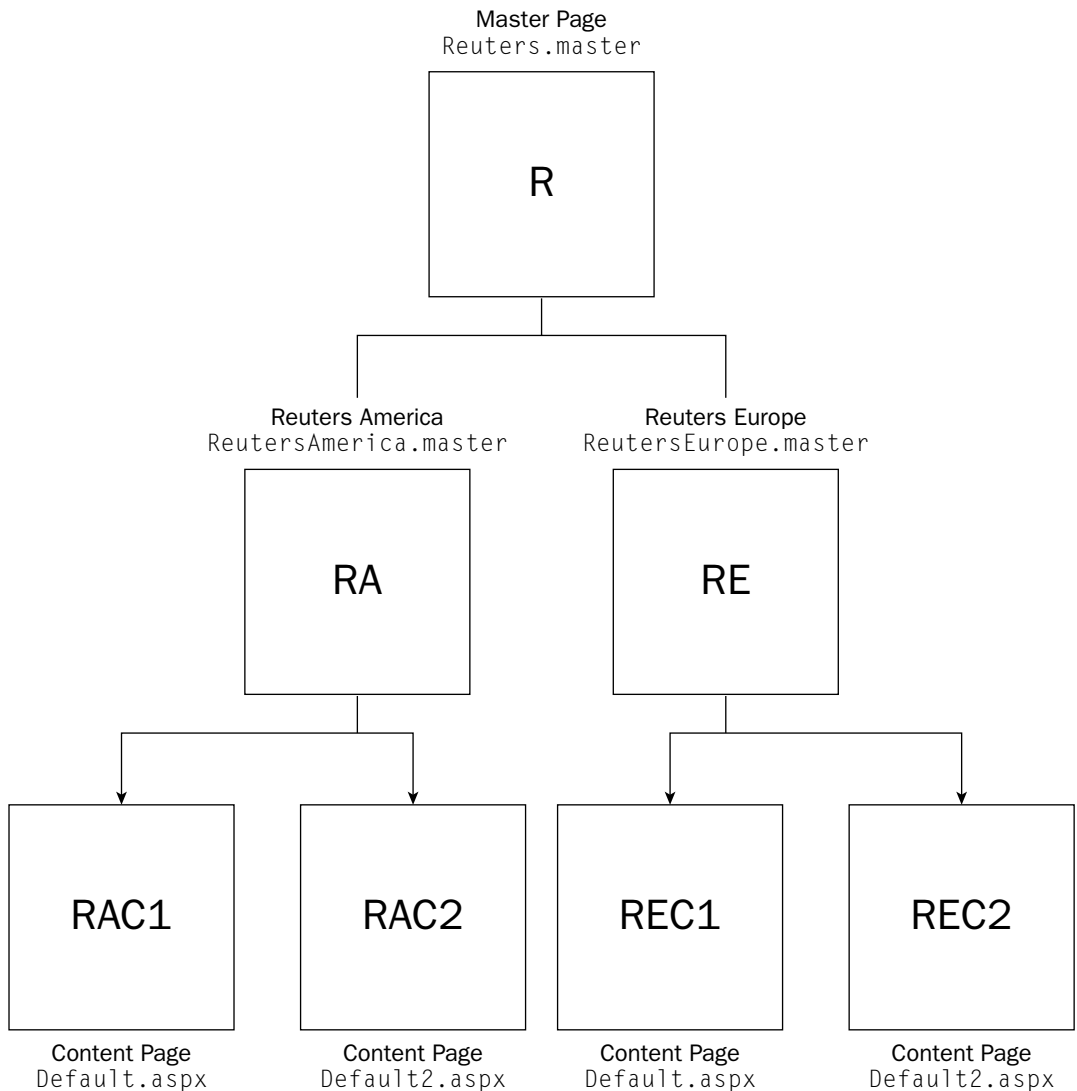


Figure 6-12

Chapter 6

The objects that you place in the content area defined with this Content control are actually placed in the defined content area within the master page. You can see this by the use of the `ContentPlaceHolderId` attribute of the Content control. This attribute is tying this content area to the content area `ContentPlaceHolder1`, which is defined in the master page.

Within this submaster page, you can also now use as many `ContentPlaceHolder` server controls as you want. Any content page that uses this master can use these controls. Listing 6-17 shows a content page that uses this submaster page, `ReutersEurope.master`.

Listing 6-17: The content page

Default.aspx

```
<%@ Page Language="VB" MasterPageFile="~/ReutersEurope.master" %>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceHolder2"
  Runat="server">
    Hello World
</asp:Content>
```

As you can see, in this content page the value of the `Master` attribute in the `Page` directive is the submaster page that you created. Inheriting this submaster page actually combines both master pages into a single master page. The Content control in this content page points to the content area defined in the submaster page as well. You can see this with the use of the `ContentPlaceHolderId` attribute. In the end, you get a very nonartistic page as shown in Figure 6-13.

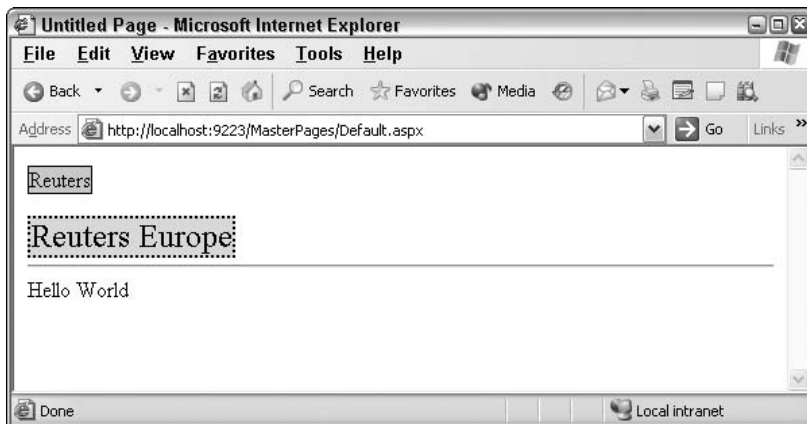


Figure 6-13

Creating a content page that uses a submaster page works pretty well. One negative point is that Visual Studio 2005 has issues with this construct, and you cannot work in the Design mode when creating your content page.

Container-Specific Master Pages

In many cases, developers are building applications that will be viewed in a multitude of different containers. Some viewers may view the application in Microsoft Internet Explorer and some might view it using Opera or Netscape Navigator. And still other viewers may call up the application on a Pocket PC or Nokia cell phone.

For this reason, ASP.NET 2.0 allows you to use multiple master pages within your content page. Depending on the viewing container used by the end user, the ASP.NET engine pulls the appropriate master file. Therefore, you want to build container-specific master pages to provide your end users with the best possible viewing experience by taking advantage of the features that a specific container provides. The capability to use multiple master pages is demonstrated in Listing 6-18.

Listing 6-18: A content page that can work with more than one master page

```
<%@ Page Language="VB" MasterPageFile="~/Wrox.master"
      Mozilla:MasterPageFile="~/WroxMozilla.master"
      Opera:MasterPageFile="~/WroxOpera.master" %>

<asp:Content ID="Content1" ContentPlaceHolderId="ContentPlaceholder1"
  Runat="server">
  Hello World
</asp:Content>
```

As you can see from this example content page, it can work with three different master page files. The first one uses the attribute `master`. This is the default setting used for any page that doesn't fit the criteria for any of the other options. This means that if the requestor is not a Mozilla or Opera browser, the default master page, `Wrox.master`, is used. However, if the requestor is an Opera browser, then `WroxOpera.master` is used instead. This is illustrated in Figure 6-14.



Figure 6-14

Chapter 6

You can find a list of available browsers on the production server where the application will be hosted at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxx\CONFIG\Browsers`. Please note that your version number may vary. Some of the available options include

- | | |
|------------------------------------|------------------------------------|
| <input type="checkbox"/> avantgo | <input type="checkbox"/> mozilla |
| <input type="checkbox"/> cassio | <input type="checkbox"/> netscape |
| <input type="checkbox"/> default | <input type="checkbox"/> nokia |
| <input type="checkbox"/> docomo | <input type="checkbox"/> openwave |
| <input type="checkbox"/> ericsson | <input type="checkbox"/> opera |
| <input type="checkbox"/> gateway | <input type="checkbox"/> palm |
| <input type="checkbox"/> generic | <input type="checkbox"/> panasonic |
| <input type="checkbox"/> goAmerica | <input type="checkbox"/> pie |
| <input type="checkbox"/> ie | <input type="checkbox"/> webtv |
| <input type="checkbox"/> jphone | <input type="checkbox"/> xiino |
| <input type="checkbox"/> MME | |

Of course, you can also add any additional `.browser` files that you deem necessary.

Event Ordering

When you work with master pages and content pages, both can use the same events (such as `Page_Load`). Be sure you know which events come before others. You are bringing two classes together to create a single page class, and a specific order is required. When an end user requests a content page in the browser, the event ordering is the following:

- ☐ **Master page child controls initialization:** All server controls contained within the master page are first initialized.
- ☐ **Content page child controls initialization:** All server controls contained in the content page are initialized.
- ☐ **Master page initialization:** The master page itself is initialized.
- ☐ **Content page initialization:** The content page is initialized.
- ☐ **Content page load:** The content page is loaded (this is the `Page_Load` event followed by the `Page_LoadComplete` event).
- ☐ **Master page load:** The master page is loaded (this is also the `Page_Load` event followed by the `Page_LoadComplete` event).
- ☐ **Master page child controls load:** The server controls on the master page are loaded onto the page.
- ☐ **Content page child controls load:** The server controls on the content page are loaded onto the page.

Pay attention to this event ordering when building your applications. If you want to use server control values that are contained on the master page within a specific content page, for example, you can't retrieve the values of these server controls from within the content page's `Page_Load` event. This is because this event is triggered before the master page's `Page_Load` event. This problem prompted the creation of the new `Page_LoadComplete` event. The content page's `Page_LoadComplete` event follows the master page's `Page_Load` event. You can, therefore, use this ordering to get at values from the master page even though it isn't populated when the content page's `Page_Load` event is triggered.

Caching with Master Pages

When working with typical `.aspx` pages, you can apply output caching to the page by using the following construct (or variation thereof):

```
<%@ OutputCache Duration="10" Varybyparam="None" %>
```

This caches the page in the server's memory for 10 seconds. Many developers use output caching to increase the performance of their ASP.NET pages. It also makes a lot of sense for use on pages with data that doesn't become stale too quickly.

How do you go about applying output caching to ASP.NET pages when working with master pages? First, you cannot apply caching to just the master page. You cannot put the `OutputCache` directive on the master page itself. If you do so, on the page's second retrieval, you get an error because the application cannot find the cached page.

To work with output caching when using a master page, stick the `OutputCache` directive in the content page. This caches both the contents of the content page as well as the contents of the master page (remember, it is just a single page at this point). The `OutputCache` directive placed in the master page does not cause the master page to produce an error, but it won't get cached. This directive only works in the content page.

Summary

When you create applications that use a common header, footer, or navigation section on pretty much every page of the application, master pages are a great solution. Master pages are easy to implement and enable you to make changes to each and every page of your application by changing a single file. Imagine how much easier this makes managing large applications that contain thousands of pages.

This chapter described master pages in ASP.NET 2.0 and explained how you build and use master pages within your Web applications. In addition to the basics, the chapter covered master page event ordering, caching, and specific master pages for specific containers. In the end, when you are working with templated applications, master pages should be your first option — the power of this approach is immense.

7

Themes and Skins

When you build a Web application, it usually has a similar look and feel across all its pages. Not too many applications are designed with each page dramatically different from the next. Generally, for your applications you use similar fonts, colors, and server control styles across all the pages.

You can apply these common styles individually to each and every server control or object on each page, or you can use a new capability provided by ASP.NET 2.0 to centrally specify these styles. All pages or parts of pages in the application can then access them.

Themes are the text-based style definitions in ASP.NET 2.0 that are the focus of this chapter.

Using ASP.NET 2.0 Packaged Themes

Themes are similar to Cascading Style Sheets (CSS) in that they enable you to define visual styles for your Web pages. Themes go further than CSS, however, in that they allow you to apply styles, graphics, and even CSS files themselves to the pages of your applications. You can apply ASP.NET themes at the application, page, or server control level.

To make life easy for the developer, ASP.NET comes with free prepackaged themes that you can use for your pages or applications. You find these themes located at `C:\WINDOWS\Microsoft .NET\Framework\v2.0.xxxxx\ASP.NETClientFiles\Themes`. The available themes that come with ASP.NET 2.0 include

- ☐ BasicBlue
- ☐ SmokeAndGlass

Applying a theme to a single ASP.NET page

In order to see how to use one of these themes, create a basic page, which includes text, a text box, a button, and a calendar. This is shown in Listing 7-1.

Listing 7-1: An ASP.NET page that does not use themes

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>INETA</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>International .NET Association (INETA)</h1><br />
        <asp:Textbox ID="TextBox1" Runat="server" /><br />
        <br />
        <asp:Calendar ID="Calendar1" Runat="server" /><br />
        <asp:Button ID="Button1" Runat="server" Text="Button" />
    </form>
</body>
</html>
```

This simple page shows some default server controls that appear just as you would expect, but that you can change with one of the ASP.NET built-in themes. When the page is called in the browser, it should look like Figure 7-1.

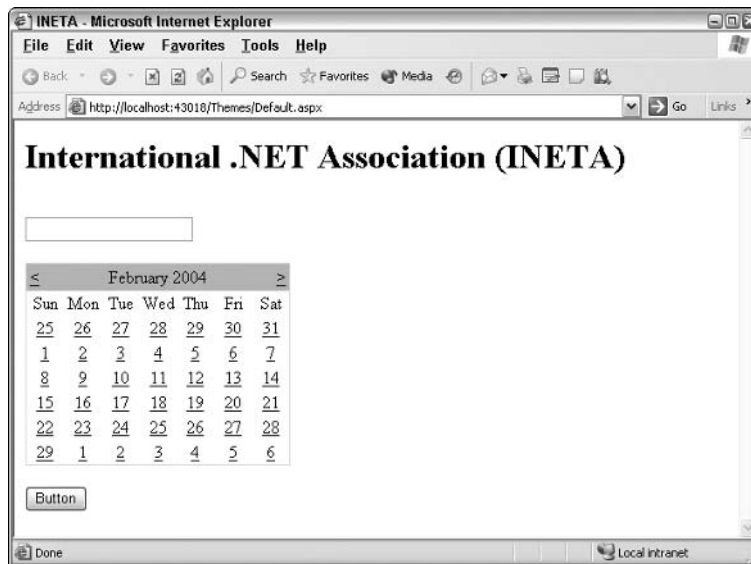


Figure 7-1

To instantly change the appearance of this page without changing the style of each server control on the page, you simply apply one of the ASP.NET default themes from within the `Page` directive:

```
<%@ Page Language="VB" Theme="SmokeAndGlass" %>
```

Adding the `Theme` attribute to the `Page` directive changes the appearance of everything on the page that is defined in the `SmokeAndGlass` theme file provided with ASP.NET 2.0. When you invoke the page in the browser, you see the result shown in Figure 7-2.

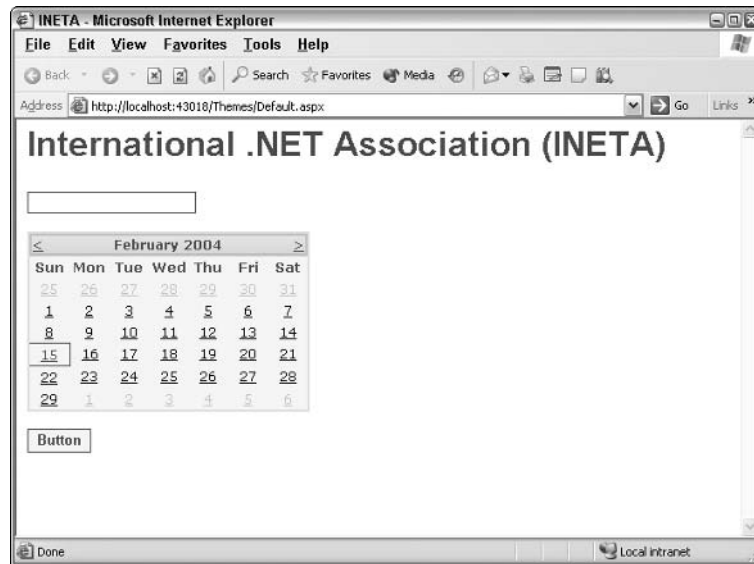


Figure 7-2

Applying a theme to an entire application

In addition to applying an ASP.NET 2.0 predefined theme to your ASP.NET pages using the `Theme` attribute within the `Page` directive, you can also apply it at an application level from the `web.config` file. This is illustrated in Listing 7-2.

Listing 7-2: Applying a theme application-wide from the `web.config` file

```
<?xml version="1.0" encoding="UTF-8" ?>

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

If you specify the theme in the `web.config` file, you don't need to define the theme again in the `Page` directive of your ASP.NET pages. This theme is applied automatically to each and every page within your application.

Applying a theme to all applications on a server

If you want to take it even one level higher, you can specify the theme that you want to use within the `machine.config` file. This is illustrated in Listing 7-3.

Listing 7-3: Specifying the theme in the machine.config file

```
<pages buffer="true" enableSessionState="true" enableViewState="true"
  enableViewStateMac="true" autoEventWireup="true" validateRequest="true"
  enablePersonalization="false" theme="SmokeAndGlass" >...</pages>
```

The `machine.config` file is located at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\CONFIG`. The `pages` node is about one-third of the way through the file. Adding the `Theme` attribute to the `pages` node within the `machine.config` file causes every Web application on that server to use the specified theme. This is a great solution if the server has multiple applications that should all be using the same theme.

If you set a theme in the `machine.config` file, you are not in any way required to use this theme for all the applications on the server. To override the theme setting placed in the `machine.config` file, you just specify another theme in the application's `web.config` file or in the Web page's `Page` directive. Remember settings that are set in the `web.config` file override settings that are in the `machine.config` file. Settings that are placed in the `Page` directive override both settings in the `machine.config` and in the `web.config` files.

Removing themes from server controls

Whether themes are set on a server, at the application level, or on a page, at times you want an alternative to the theme that has been defined. For example, change the text box server control that you have been working with (from Listing 7-1) by making its background black and using white text:

```
<asp:Textbox ID="TextBox1" Runat="server"
  BackColor="#000000" ForeColor="#ffffff" />
```

The black background color and the color of the text in the text box are specified directly in the control itself with the use of the `BackColor` and `ForeColor` attributes. If you have applied a theme to the page where this text box control is located, however, you won't see this black background or white text because these changes are overridden by the theme itself.

To apply a theme to your ASP.NET page but not to this text box control, you simply use the `EnableTheming` property of the text box server control:

```
<asp:Textbox ID="TextBox1" Runat="server"
  BackColor="#000000" ForeColor="#ffffff" EnableTheming="false" />
```

If you apply this control to the text box server control from Listing 7-1 with the `SmokeAndGlass` theme applied to the entire page, the theme is applied to every control on the page *except* the text box. This result is shown in Figure 7-3.

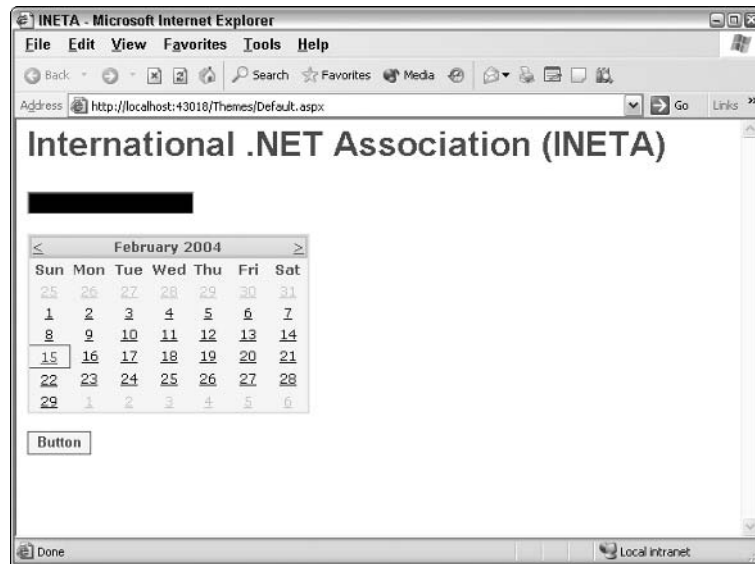


Figure 7-3

If you want to turn off theming for multiple controls within a page, consider using the Panel control to encapsulate a collection of controls and then set the `EnableTheming` attribute of the Panel control to `False`. This disables theming for each control contained within the Panel control.

Removing themes from Web pages

Now what if, when you set the theme for an entire application in the `web.config` file, you want to exclude a single ASP.NET page? It is quite possible to remove a theme setting at the page level, just as it is at the server control level.

The `Page` directive includes an `EnableTheming` attribute that can be used to remove theming from your ASP.NET pages. To remove the theme that would be applied by the theme setting in the `web.config` or `machine.config` file, you simply construct your `Page` directive in the following manner:

```
<%@ Page Language="VB" EnableTheming="False" %>
```

This construct sets the theme to nothing — thereby removing any settings that were specified in the `web.config` or `machine.config` files.

Removing themes from applications

Because themes can be set in the `machine.config` file that affect every application on the server, you might sometimes want to remove the theme setting from the application that you are working on. To do this, you specify no theme in the `web.config` file. This construct is shown in Listing 7-4.

Listing 7-4: Removing the server-set theme in the `web.config` file

```
<?xml version="1.0" encoding="UTF-8" ?>

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

Creating Your Own Themes

When you are applying themes to your applications, you are in no way limited just to default themes provided with ASP.NET. You can easily create your own themes. The themes that you create can be applied at the application level or put in the server theme repository along with the Microsoft default themes that come with the ASP.NET 2.0 install. As you can see, themes are a great way to easily apply a consistent look and feel across your entire application.

Creating the proper folder structure

In order to create your own themes for an application, you first need to create the proper folder structure in your application. To do this, right-click your project and add a new folder. Name the folder `Themes`. Notice when you do this that the `Themes` folder does not have the typical folder icon next to it, but instead has a folder icon that includes a paint brush. This is shown in Figure 7-4.

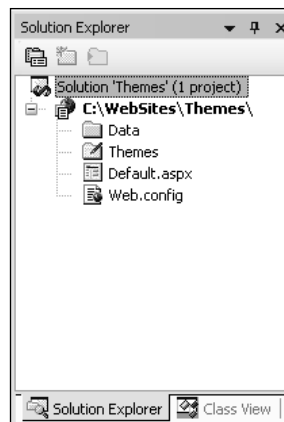


Figure 7-4

Within the `Themes` folder itself, you create an additional theme folder for each and every theme that you might use in your application. For instance, if you are going to have four themes — Summer, Fall, Winter, and Spring — then you create four folders that are named appropriately.

You might use more than one theme in your application for many reasons — season changes, day/night changes, category of user, or even user preferences.

Each theme folder must contain the elements of the theme, which can include

- ❑ A single skin file
- ❑ CSS files
- ❑ Images

Creating a skin

A *skin* is a definition of styles applied to server controls in your ASP.NET page. Skins can work in conjunction with CSS files or images. To create a theme to use in your ASP.NET applications, you use just a single skin file in the theme folder. The skin file can have any name, but it must have a `.skin` file extension.

Even though you have four theme folders in your application, concentrate on the creation of the Summer theme for the purposes of this chapter. Within the `Summer` folder in your project, create a text file called `Summer.skin`. If you try to right-click the `Summer` theme folder and select `Add New Item`, notice that a skin file isn't listed among the options. Therefore, select the `Text File` option and name the file `Summer.skin`. Then create a skin file as shown in Listing 7-5.

Listing 7-5: The `Summer.skin` file

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
           Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
             Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
             BorderColor="#004000" Font-Bold="True" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

This is just a sampling of what the `Summer.skin` file should be. If you are going to use it in a real application, you actually make a definition for each and every server control option. In this case, you have a definition in place for three different types of server controls — the `Label`, `TextBox`, and `Button` controls. After saving the `Summer.skin` file in the `Summer` folder, your file structure should resemble Figure 7-5 from the Solution Explorer of Visual Studio 2005.

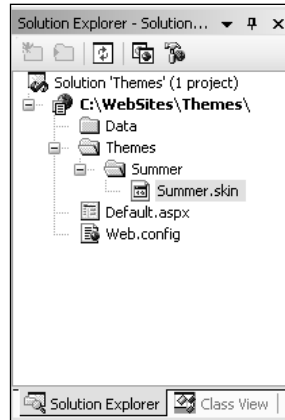


Figure 7-5

Just like the regular server control definitions that you put on a typical .aspx page, these control definitions must contain the `Runat="server"` attribute. If you specify this attribute in the skinned version of the control, you also include it in the server control you put on an .aspx page that uses this theme. Also notice is that no `ID` attribute is specified in the skinned version of the control. If you specify an `ID` attribute here, you get an error when a page tries to use this theme.

As you can see, you supply a lot of different visual definitions to these three controls and this should give the page a summery look and feel. An ASP.NET page in this project can simply use this custom theme as it would any global Microsoft pre-installed theme (see Listing 7-6).

Listing 7-6: Using the Summer theme in an ASP.NET page

VB

```
<%@ Page Language="VB" Theme="Summer" %>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & TextBox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>INETA</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Textbox ID="TextBox1" Runat="server">
        </asp:Textbox>
        <br />
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit Your Name"
            OnClick="Button1_Click" />
        <br />
    </form>
</body>
</html>
```

```

        <br />
        <asp:Label ID="Label1" Runat="server" />
    </form>
</body>
</html>

```

C#

```

<%@ Page Language="C#" Theme="Summer" %>

<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text.ToString();
    }
</script>

```

As you can see from the server controls on this .aspx page, no styles are associated with them. These are just the default server controls that you drag and drop onto the design surface of Visual Studio 2005. There is, however, the style that you defined in the Summer.skin file, as shown in Figure 7-6.

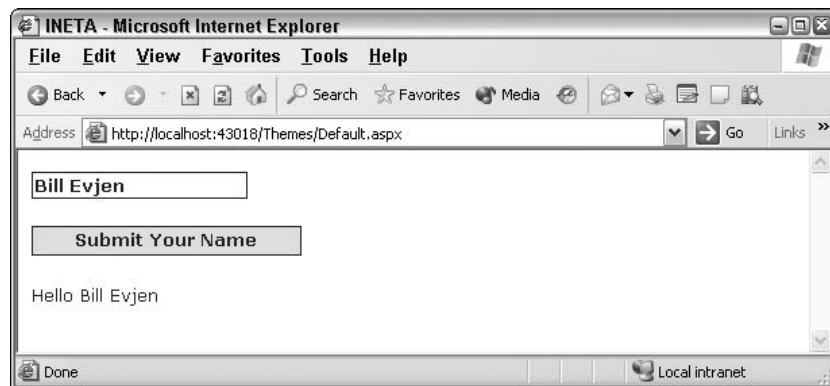


Figure 7-6

Including CSS files in your themes

In addition to the server control definitions that you create from within a .skin file, you can make further definitions using Cascading Style Sheets (CSS). You might have noticed, when using a .skin file, that you could define only the styles associated with server controls and nothing else. But developers usually use quite a bit more than server controls in their ASP.NET pages. For instance, ASP.NET pages are routinely made up of HTML server controls, raw HTML, or even raw text. As the Summer theme stands at present it has only a Summer.skin file associated with it. Any of these other items would have no style whatsoever applied to them.

For a theme that goes beyond the server controls, you must further define the theme style so that HTML server controls, HTML, and raw text are all changed according to the theme. You achieve this with a CSS file within your Themes folder.

Chapter 7

It is rather easy to create CSS files for your themes when using Visual Studio 2005. Right-click the `Summer` theme folder and select `Add New Item`. In the list of options, select the option `Style Sheet` and name it `Summer.css`. The `Summer.css` file should be sitting right next to your `Summer.skin` file. This creates an empty `.css` file for your theme. I won't go into the details of how to make a CSS file using Visual Studio 2005 and the CSS creation tool. The process is the same as in previous versions. I just want to point out that it is quite simple to do this because the dialog that comes with Visual Studio 2005 enables you to completely define your CSS page with no need to code. A sample dialog is shown in Figure 7-7.

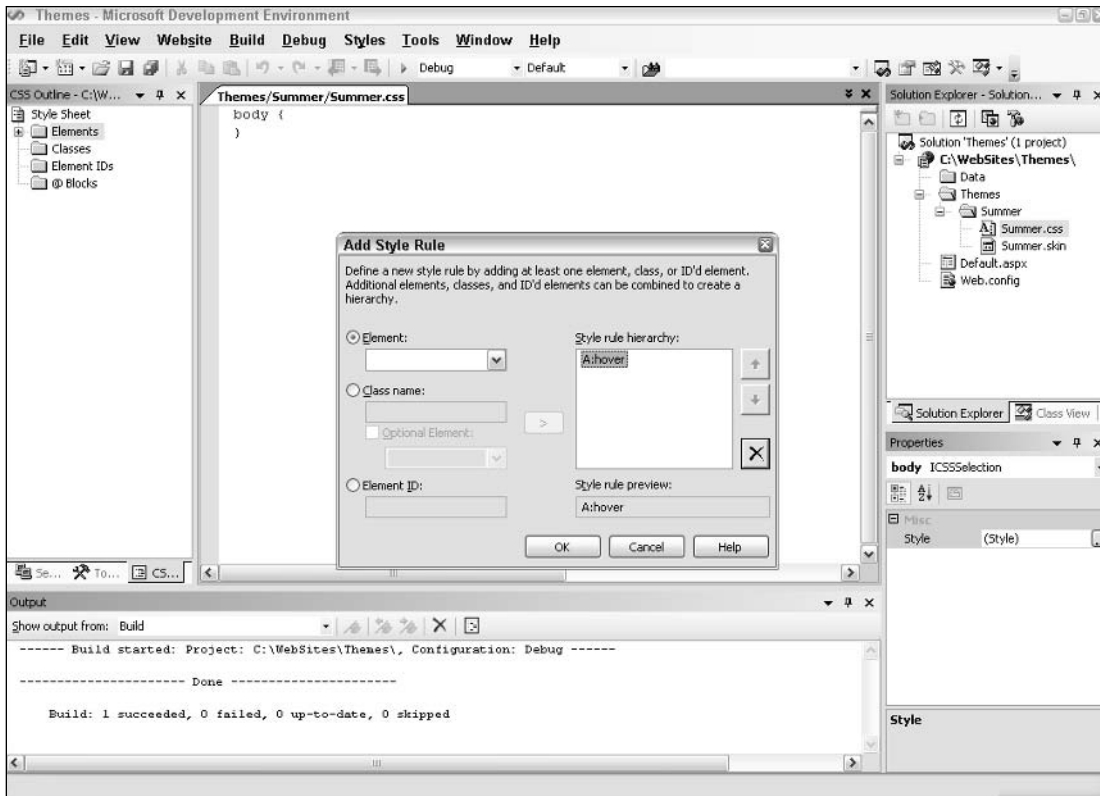


Figure 7-7

To create a comprehensive theme with this dialog, you define each HTML element that might appear in the ASP.NET page. This can be a lot of work, but it's worth it in the end. For now, create a small CSS file that changes some of the nonserver control items on your ASP.NET page. This CSS file is shown in Listing 7-7.

Listing 7-7: A CSS file with some definitions

```
body
{
    font-size: x-small;
```

```
font-family: Verdana;
color: #004000;
}

A:link {
    color: Blue;
    text-decoration:none;
}

A:visited
{
    color: Blue;
    text-decoration:none;
}

A:hover {
    COLOR: Red;
    text-decoration:underline overline;
}
```

Suppose that there is a definition for the TextBox server control in the .skin file:

```
<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
  BackColor="#ffffff" Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
  BorderColor="#004000" Font-Bold="True" />
```

But, what if you also had a definition in your .css file for each <input> element in the ASP.NET page as shown here:

```
INPUT
{
  background-color: black;
}
```

When you run the .aspx page with these kinds of style conflicts, the .skin file takes precedence over styles applied to every HTML element that is created using ASP.NET server controls regardless of what the .css file says. In fact, this sort of scenario gives you a page in which the <input> element that is created from the server control is white as defined in the .skin file and the second text box is black as defined in the .css file. This is shown in Figure 7-8.

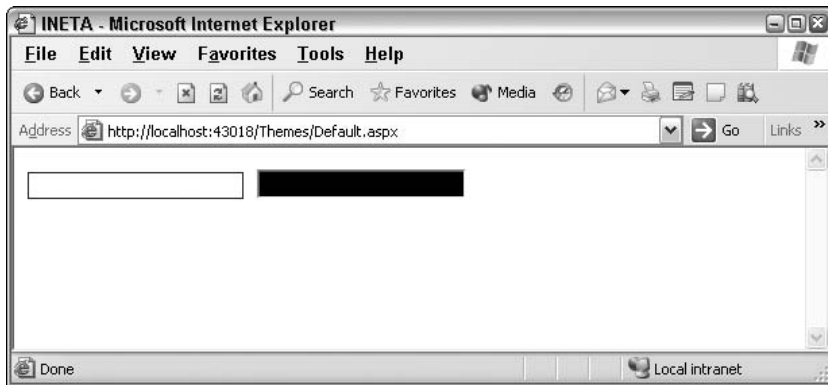


Figure 7-8

Having your themes include images

Probably one of the coolest reasons why themes, rather than CSS, are the better approach for applying a consistent style to your Web page is that themes enable you to incorporate actual images into the style definitions.

A lot of controls use images to create a better visual appearance. The first step in incorporating images into your server controls that consistently use themes is to create an `Images` folder within the `Themes` folder itself, as illustrated in Figure 7-9.

You have a couple of easy ways to use the images that you might place in this folder. The first is to incorporate the images directly from the .skin file itself. You can do this with the `TreeView` server control. The `TreeView` control can contain images used to open and close nodes for navigation purposes. You can place images in your theme for each and every `TreeView` control in your application. If you do that, you can then define the `TreeView` server control in the .skin file, as shown in Listing 7-8.

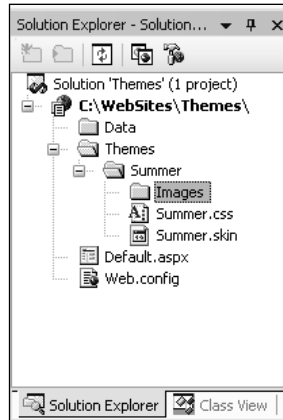


Figure 7-9

Listing 7-8: Using images from the theme folder in a TreeView server control

```
<asp:TreeView runat="server" BorderColor="#FFFFFF" BackColor="#FFFFFF"
  ForeColor="#585880" Font-Size=".9em" Font-Names="Verdana"
  LeafNodeImageUrl="images\summer_iconlevel.gif"
  RootNodeImageUrl="images\summer_iconmain.gif"
  ParentNodeImageUrl="images\summer_iconmain.gif" NodeIndent="30"
  CollapseImageUrl="images\summer_minus.gif"
  ExpandImageUrl="images\summer_plus.gif">
  ...
</asp:TreeView>
```

When you run a page containing a TreeView server control, it is populated with the images held in the Images folder of the theme.

It is easy to incorporate images into the TreeView control. It even specifically asks for an image location as an attribute of the control. The new WebParts controls are used to build portals. Listing 7-9 is an example of a Web Part definition from a .skin file that incorporates images from the Images folder of the theme.

Listing 7-9: Using images from the theme folder in a WebPartZone server control

```
<asp:WebPartZone runat="server" PartFrameType="TitleAndBorder"
  DragHighlightColor="#6464FE" ShowIconInPartTitle="True" BorderStyle="double"
  BorderColor="#E7E5DB" BorderWidth="2pt" BackColor="#F8F8FC"
  cssclass="theme_fadeblue" Font-Size=".9em" Font-Names="Verdana">
  <PartContentStyle ForeColor="#585880" BorderStyle="double"
    BorderColor="#585880" BorderWidth="1pt"
    BackColor="#FFFFFF"></PartContentStyle>
  <FooterStyle ForeColor="#585880" BackColor="#CCCCC"></FooterStyle>
  <WebPartHelpVerb ImageURL="images/SmokeAndGlass_help.gif"
    checked="False" enabled="True" visible="True"></WebPartHelpVerb>
  <WebPartCloseVerb ImageURL="images/SmokeAndGlass_close.gif"
    checked="False" enabled="True" visible="True"></WebPartCloseVerb>
  <WebPartRestoreVerb ImageURL="images/SmokeAndGlass_restore.gif">
```



```
checked="False" enabled="True" visible="True"></WebPartRestoreVerb>
<WebPartMinimizeVerb ImageURL="images/SmokeAndGlass_minimize.gif"
checked="False" enabled="True" visible="True"></WebPartMinimizeVerb>
<WebPartEditVerb ImageURL="images/SmokeAndGlass_edit.gif"
checked="False" enabled="True" visible="True"></WebPartEditVerb>
<TitleStyle ForeColor="#FFFFFF" Font-Names="Verdana"
BorderStyle="double" BorderWidth="0" Font-Bold="true" BorderColor="#E7E5DB"
BackColor="#232377"></TitleStyle>
<PartStyle ForeColor="#585880" Font-Names="Verdana" Font-Size=".9em"
BorderColor="#44448A" BackColor="#F8F7F4"></PartStyle>
<PartTitleStyle ForeColor="#585880" Font-Names="Verdana"
BorderStyle="solid" Font-Bold="true" BorderWidth="1pt" Font-Size=".9em"
BorderColor="#494979" BackColor="#F8F7F4"
cssclass="theme_header"></PartTitleStyle>
<PartVerbStyle ForeColor="#FFFFFF" Font-Names="Verdana" Font-
Underline="False" Font-Size=".7em" BorderColor="#000066" BorderWidth="1pt"
BackColor="#8383B6"></PartVerbStyle>
<EditWebPartStyle ForeColor="#6464FE" BorderColor="#6464FE"
BackColor="#6464FE" Font-Size=".9em" Font-Names="Verdana" />
</asp:WebPartZone>
```

As you can see here, this series of toolbar buttons that are in a WebPart now use images that come from the SmokeAndGlass theme. When this WebPart is generated, the style is defined directly from the .skin file, but the images specified in the .skin file are retrieved from the Images folder in the theme itself.

Not all server controls enable you to work with images directly from the Themes folder by giving you an image attribute to work with. If you don't have this capability, you must work with the .skin file and the CSS file together. If you do, you can place your theme-based images in any element you want. The SmokeAndGlass theme that comes with ASP.NET 2.0 is a good example of how to do this.

Place the image that you want to use in the Images folder just as you normally would. Then define the use of the images in the .css file. The SmokeAndGlass example in Listing 7-10 demonstrates this.

Listing 7-10: Part of the CSS file from SmokeAndGlass.css

```
.theme_header {
background-image :url( images/smokeandglass_brownfadetop.gif);
}

.theme_highlighted {
background-image :url( images/smokeandglass_blueandwhitef.gif);
}

.theme_fadeblue {
background-image :url( images/smokeandglass_fadeblue.gif);
}
```

These are not styles for a specific HTML element; instead, they are CSS classes that you can put into any HTML element that you want. In this case, each CSS class mentioned here is defining a specific background image to use for the element.

After it is defined in the CSS file, you can utilize this CSS class in the .skin file when defining your server controls. Listing 7-11 shows you how.

Listing 7-11: Using the CSS class in one of the server controls defined in the .skin file

```
<asp:Calendar runat="server" BorderStyle="double" BorderColor="#E7E5DB"
  BorderWidth="2" BackColor="#F8F7F4" Font-Size=".9em" Font-Names="Verdana">
  <TodayDayStyle      BackColor="#F8F7F4" BorderWidth="1" BorderColor="#585880"
    ForeColor="#585880" />
  <OtherMonthDayStyle BackColor="transparent" ForeColor="#CCCCCC" />
  <SelectedDayStyle   ForeColor="#6464FE" BackColor="transparent"
    cssclass="theme_highlighted" />
  <TitleStyle         Font-Bold="True"   BackColor="#CCCCCC" ForeColor="#585880"
    BorderColor="#CCCCCC" BorderWidth="1pt" cssclass="theme_header" />
  <NextPrevStyle      Font-Bold="True"   ForeColor="#585880"
    BorderColor="transparent" BackColor="transparent" />
  <DayStyle            ForeColor="#000000"
    BorderColor="transparent" BackColor="transparent" />
  <SelectorStyle      Font-Bold="True"   ForeColor="#696969" BackColor="#F8F7F4"
    />
  <WeekendDayStyle    Font-Bold="False"  ForeColor="#000000"
    BackColor="transparent" />
  <DayHeaderStyle     Font-Bold="True"   ForeColor="#585880"
    BackColor="Transparent" />
</asp:Calendar>
```

This Calendar server control definition from a .skin file uses one of the earlier CSS classes in its definition. It actually uses an image that is specified in the CSS file in two different spots within the control (shown in bold). It is first specified in the <SelectedDayStyle> element. Here you see the attribute and value **cssclass="theme_highlighted"**. The other spot is within the <TitleStyle> element. In this case, it is using **theme_header**. When the control is rendered, these CSS classes are referenced and finally point to the images that are defined in the CSS file.

It is interesting that the images used here for the header of the Calendar control don't really have much to them. For instance, the `smokeandglass_brownfadetop.gif` image is simply a thin, gray sliver, as shown in Figure 7-10.



Figure 7-10

This very small image (in this case, very thin) is actually repeated as often as necessary to make it equal the length of the header in the Calendar control. The image is lighter at the top and darkens toward the bottom. Repeated horizontally, any control like this gives a three-dimensional effect to the control. Try it out, and you get the result shown in Figure 7-11.

< February 2004 >						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	1	2	3	4	5	6

Figure 7-11

Defining Multiple Skin Options

Using the themes technology in ASP.NET 2.0, you can have a single theme; but also, within the theme's .skin file, you can have specific controls that are defined in multiple ways. You can frequently take advantage of this feature within your themes. For instance, you might have text box elements scattered throughout your application, but you might not want each and every text box to have the same visual appearance. In this case, you can create multiple versions of the <asp:textbox> server control within your .skin file. In Listing 7-12 you see how to create multiple versions of the <asp:textbox> control in the .skin file from Listing 7-5.

Listing 7-12: The Summer.skin file, which contains multiple version of the <asp:textbox> server control

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
    Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
    Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
    BorderColor="#004000" Font-Bold="True" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"
    Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
    BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Arial"
    Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
    BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
    Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
    BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

In this .skin file, you can see three definitions in place for the TextBox server control. The first one is the same as before. Although the second and third definitions have a different style, they also contain a new attribute in the definition — *SkinID*. To create multiple definitions of a single element, you use the *SkinID* attribute to differentiate among the definitions. The value used in the *SkinID* can be anything you want. In this case, it is *TextboxDotted* and *TextboxDashed*.

Note that no `SkinID` attribute is used for the first `<asp:Textbox>` definition. By not using one, you are saying that for each `<asp:Textbox>` control on an ASP.NET page that uses this theme but has no pointer to a `SkinID`, this is the default style definition to use.

Take a look at a sample `.aspx` page that uses this `.skin` file, Listing 7-13.

Listing 7-13: A simple `.aspx` page that uses the `Summer.skin` file with multiple text-box style definitions

```
<%@ Page Language="VB" Theme="Summer" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Different SkinIDs</title>
</head>
<body>
  <form id="form1" runat="server">
    <p>
      <asp:Textbox ID="TextBox1" Runat="server">Textbox1</asp:Textbox>
    </p><p>
      <asp:Textbox ID="TextBox2" Runat="server"
        SkinId="TextboxDotted">Textbox2</asp:Textbox>
    </p><p>
      <asp:Textbox ID="TextBox3" Runat="server"
        SkinId="TextboxDashed">Textbox3</asp:Textbox>
    </p>
  </form>
</body>
</html>
```

This small `.aspx` page shows three text boxes, each of a different style. When you run this page, you get the results shown in Figure 7-12.

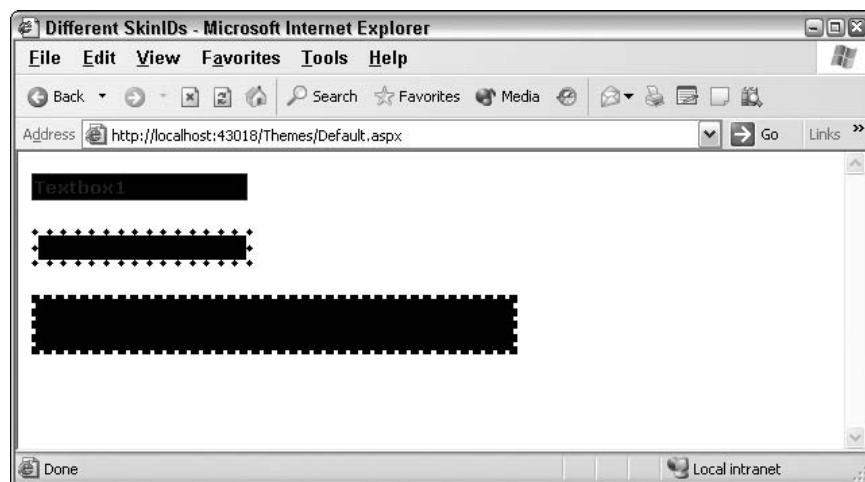


Figure 7-12

The first text box doesn't point to any particular `SkinID` in the `.skin` file. Therefore, the default skin is used. As stated before, the default skin is the one in the `.skin` file that doesn't have a `SkinID` attribute in it. The second text box then contains `skinid="TextboxDotted"` and, therefore, inherits the style definition defined in the `TextboxDotted` skin in the `Summer.skin` file. The third text box takes the `SkinID` `TextboxDashed` and is also changed appropriately.

As you can see, it is quite simple to define multiple versions of a control that can be used throughout your entire application.

Programmatically Working with Themes

So far, you have seen examples of working with ASP.NET 2.0 themes in a declarative fashion, but you can also work with themes programmatically.

Assigning the page's theme programmatically

To programmatically assign the theme to the page, use the construct shown in Listing 7-14.

Listing 7-14: Assigning the theme of the page programmatically

VB

```
<script runat="server" language="vb">
    Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
        Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    void Page_PreInit(object sender, System.EventArgs e)
    {
        Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

You must set the `Theme` of the `Page` property in or before the `Page_PreInit` event for any static controls that are on the page. If you are working with dynamic controls, set the `Theme` property before adding it to the `Controls` collection.

Assigning a control's SkinID programmatically

Another option is to assign a specific server control's `SkinID` property programmatically (see Listing 7-15).

Listing 7-15: Assigning the server control's SkinID property programmatically

VB

```
<script runat="server" language="vb">
    Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
```

```

        TextBox1.SkinID = "TextboxDashed"
    End Sub
</script>

```

C#

```

<script runat="server">
    void Page_PreInit(object sender, System.EventArgs e)
    {
        TextBox1.SkinID = "TextboxDashed";
    }
</script>

```

Again, you assign this property before or in the `Page_PreInit` event in your code.

Themes and Custom Controls

If you are building custom controls in an ASP.NET 2.0 world, understand that end users can also apply themes to the controls that they use in their pages. By default, your custom controls are theme enabled whether your custom control inherits from `Control` or `WebControl`.

To disable theming for your control, you can simply use the `EnableTheming` attribute on your class. This is illustrated in Listing 7-16.

Listing 7-16: Disabling theming for your custom controls

VB

```

Namespace Wrox.ServerControls

    <EnableTheming(False)> _
    Public Class SimpleHello
        Inherits System.Web.UI.Control

        Private _name As String

        Public Property Name() As String
            Get
                Return _name
            End Get
            Set(ByVal Value As String)
                _name = Value
            End Set
        End Property

        Protected Overrides Sub RenderContents(ByVal controlOutput As _
            HtmlTextWriter)
            controlOutput.Write("Hello " + Name)
        End Sub

    End Class

End Namespace

```

(continued)

Listing 7-16: *(continued)***C#**

```
namespace Wrox.ServerControls
{
    [EnableTheming(false)]
    public class SimpleHello : Control
    {
        private string _name;

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        protected override void RenderContents (HtmlTextWriter controlOutput)
        {
            controlOutput.Write ("Hello " + Name);
        }
    }
}
```

You can also disable theming for the individual properties that might be in your custom controls. This is done as illustrated in Listing 7-17.

Listing 7-17: Disabling theming for properties in your custom controls**VB**

```
Namespace Wrox.ServerControls

    Public Class SimpleHello
        Inherits System.Web.UI.Control

        Private _myValue As String

        <Themeable(False)>
        Public Property MyCustomProperty() As String
            Get
                Return _myValue
            End Get
            Set(ByVal Value As String)
                _myValue = Value
            End Set
        End Property

    End Class

End Namespace
```

```
C#
namespace Wrox.ServerControls
{
    public class SimpleHello : Control
    {
        private string _myValue;

        [Themeable(false)]
        public string Name
        {
            get { return _myValue; }
            set { _myValue = value; }
        }
    }
}
```

Summary

With the addition of themes and skins in ASP.NET 2.0, it has become quite easy to apply a consistent look and feel across your entire application. Remember that themes can just contain simple server control definitions in a `.skin` file or elaborate style definitions, which include not only `.skin` files, but also CSS style definitions and even images!

As you will see later in the book, you can use themes in conjunction with the new personalization features that ASP.NET 2.0 provides. This can enable your end users to customize their experiences by selecting their own themes. Your application can present a theme just for them, and it can remember their choices through the APIs that are offered in ASP.NET 2.0.

8

Membership and Role Management

The authentication and authorization of users are important functions in many Web sites and browser-based applications. Traditionally, when working with Microsoft's Windows Forms applications (thick-client), you depended on Windows Integrated Authentication; when working with browser-based applications (thin-client), you used forms authentication.

Forms authentication enabled you to take requests that were not yet authenticated and redirect them to an HTML form using HTTP client-side redirection. The user provided his login information and submitted the form. After the application authenticated the request, the user received an HTTP cookie, which was then used on any subsequent requests. This kind of authentication was fine in many ways, but it required developers to build every element and even manage the back-end mechanics of the overall system. This was a daunting task for many developers and, in most cases, it was rather time-consuming.

ASP.NET 2.0 introduces a new authentication and authorization management service that takes care of the login, authentication, authorization, and management of users who require access to your Web pages or applications. This outstanding new Membership and Role Management Service is an easy-to-implement framework that works out of the box using either Microsoft Access or Microsoft SQL Server as the back-end data store. This new framework also includes a new API that allows for programmatic access to the capabilities of both the membership and role management services. In addition, a number of new server controls make it easy to create Web applications that incorporate everything these services have to offer.

Before you look at the new membership and role management features of ASP.NET 2.0, here's a quick review of authentication and authorization.

Authentication

Authentication is a process that determines the identity of a user. After a user has been authenticated, a developer can determine if the identified user has *authorization* to proceed. It is impossible to give an entity authorization if no authentication process has been applied. Authentication is provided in ASP.NET 2.0 through the use of the new membership service.

Authorization

Authorization is the process determining whether an authenticated user is allowed access to any part of an application, access to specific points of an application, or access only to specific datasets that the application provides. Authenticating and authorizing users or groups enable you to customize a site based on user types or preferences. Authorization is provided in ASP.NET 2.0 through the use of a new role management service.

ASP.NET 2.0 Authentication

ASP.NET 2.0 provides the membership management service to deal with authenticating users to access a page or an entire site. The new ASP.NET management service not only provides a new API suite for managing users, but it also gives you some new server controls. These new server controls work with the end user through the process of authentication. Shortly, you will look at the functionality of these controls.

Setting up your Web site for membership

Before you can use the security controls that are provided with ASP.NET 2.0, you first have to set up your application to work with the new membership service. How you do this depends on how you approach the security framework provided.

By default, ASP.NET 2.0 uses the built-in `AspNetAccessProvider` for storing details about the registered users of your application. Also, for the initial demonstrations, you will work with Forms authentication. I assume that the application is open to the public for registration and viewing. If it were an intranet-based application (meaning that all the users are on a particular network), then you would use Windows Integrated Authentication for authenticating users.

ASP.NET 2.0, as you know, offers a data provider model that handles the detailed management required to interact with multiple types of underlying data stores. Figure 8-1 shows a diagram of the new ASP.NET 2.0 membership service.

From the diagram, you can see that like the rest of the ASP.NET 2.0 provider models, the membership providers can access a wide variety of underlying data stores. In this diagram, you can see the built-in Microsoft Access and Microsoft SQL Server data stores. You can also build your own membership providers to get at any other custom data stores that work with user credentials. Above the membership providers in the diagram, you can see a list of security server controls that utilize the access granted by providers to work with the users in the authentication process.

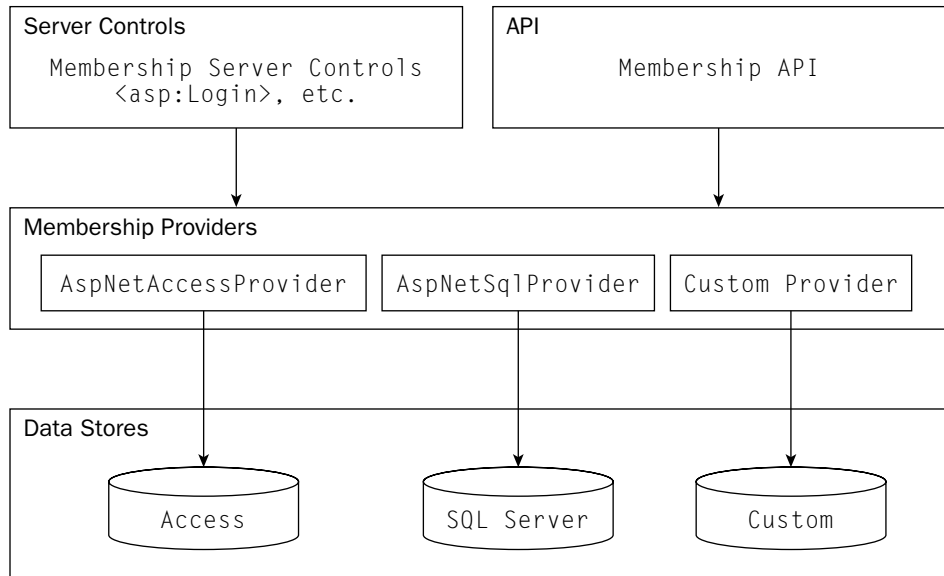


Figure 8-1

Adding an <authentication> element to the web.config file

To allow forms authentication in your Web application for the new membership service, the first step is to turn on this feature from the `web.config` file. So create a `web.config` file if you don't already have one. Then, add the section shown in Listing 8-1 to the file.

Listing 8-1: Adding Forms authentication to the web.config file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
  </system.web>
</configuration>
```

The simple addition of the <authentication> element to the `web.config` file turns on everything that you need to start using the membership service provided by ASP.NET 2.0. To turn on the forms authentication using this element, you simply give the value `Forms` to the `mode` attribute. This is a forms authentication example, but other possible values of the `mode` attribute include `Windows`, `Passport`, or `None`.

IIS authentication schemes include basic, digest, and Integrated Windows Authentication. Passport authentication points to a centralized service provided by Microsoft that offers a single login and core profile service for any member sites (it costs money to use Passport).

Because the `mode` attribute in our example is set to `Forms`, you can move on to the next step of adding users to the data store. You can also change the behavior of the forms authentication system at this point by making some modifications to the `web.config` file. These possibilities are reviewed next.

Adding a <forms> element to the web.config file

Using forms authentication, you can provide users with access to a site or materials based upon credentials they input into a Web-based form. When an end user attempts to access a Web site, he is entering the site using anonymous authentication, which is the default authentication mode. If he is found to be anonymous, he can be redirected (by ASP.NET) to a specified login page. After the end user passes the authentication process, he is provided with an HTTP cookie, which can be used in any subsequent requests.

You can see the possibilities of the forms authentication setting in Listing 8-2, which shows possible changes to the web.config file.

Listing 8-2: Modifying the Forms authentication behavior

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ASPXAUTH"
        loginUrl="login.aspx"
        protection="All"
        timeout="30"
        path="/"
        requireSSL="false"
        slidingExpiration="true"
        cookieless="useDeviceProfile"
      </forms>
    </authentication>
  </system.web>
</configuration>
```

You can set these as you wish, and you have plenty of options for values other than the ones that are displayed. Also, as I stated earlier, these values are not required. You can use the membership service right away with only what was shown in Listing 8-1.

You can find some interesting settings in Listing 8-2, however. You can change how the forms authentication system works by adding a <forms> element to the web.config file. Make sure that you have the <forms> element nested within the <authentication> elements. The following list describes the possible attributes of the <forms> element:

- ❑ **name:** Defines the name used for the cookie sent to the end users after they have been authenticated. By default, this cookie is named .ASPXAUTH.
- ❑ **loginUrl:** Specifies the page location to which the HTTP request is redirected for login if no valid authentication cookie (.ASPXAUTH or otherwise) is found. By default, it is set to login.aspx.
- ❑ **protection:** Specifies the amount of protection that you want to apply to the cookie that is stored on the end user's machine after he has been authenticated. The possible settings include All, None, Encryption, and Validation. You should always attempt to use All.
- ❑ **timeout:** Defines the amount of time (in minutes) after which the cookie expires. The default value is 30 minutes.

- ❑ `path`: Specifies the path for cookies issued by the application.
- ❑ `requireSSL`: Defines whether you require that credentials be sent over an encrypted wire (SSL) instead of clear text.
- ❑ `slidingExpiration`: Specifies whether the timeout of the cookie is on a sliding scale. The default value is `True`. This means that the end user's cookie does not expire until 30 minutes (or the time specified in the `timeout` attribute) after the last request to the application has been made. If the value of the `slidingExpiration` attribute is set to `False`, the cookie expires 30 minutes from the first request.
- ❑ `cookieless`: Specifies how the cookies are handled by ASP.NET. The possible values include `useDeviceProfile`, `useCookies`, `auto`, and `useUri`. The default value is `useDeviceProfile`. This value detects whether to use cookies based on the user agent of the device. `useCookies` requires that all requests have the credentials stored in a cookie. `auto` auto-determines whether the details are stored in a cookie on the client or within the URI (this is done by sending a test cookie first). Finally, `useUri` forces ASP.NET to store the details within the URI on all instances.

Now that forms authentication is turned on, the next step is adding users to the Microsoft Access data store.

Adding users

To add users to the membership service, you can register users into the Microsoft Access data store. The first question you might ask is, "Where is this data store?"

The Microsoft Access provider uses an Access file structured specifically for the membership service (and other ASP.NET systems). You can find a templated version of this Access file at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\ASPNetdb_Template.mdb`. One option is to make a copy of this file and place the copy in the Data folder of your solution. If you take this approach, be sure to rename the file `AspNetDB.mdb`.

The other option is to let Visual Studio 2005 create it for you. To do this, you work with the ASP.NET server controls that utilize the membership service to force the creation of this file (as I explain later).

Now that the data store is in place (or you let Visual Studio take care of it for you), it is time to start adding users to the data store.

Using the CreateUserWizard server control

The first server control that utilizes the membership service is the `CreateUserWizard` server control. This control enables you to plug registered users into your data store for later retrieval. If the first page of your application allows end users to register for your site, you want, at a minimum, to retrieve a login and password from the user so that he can use these items later to log in to the site.

To make your life as simple as possible, the `CreateUserWizard` control takes complete control of doing all these things. Listing 8-3 shows a simple use of the control.

Listing 8-3: Allowing end users to register with the site

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Creating Users</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:CreateUserWizard ID="CreateUserWizard1" Runat="server"
            BorderWidth="1px" BorderColor="#FFDFAD" BorderStyle="Solid"
            BackColor="#FFFBD6" Font-Names="Verdana" Font-Size="0.8em">
            <TitleTextStyle Font-Bold="True" BackColor="#990000"
                ForeColor="White"></TitleTextStyle>
        </asp:CreateUserWizard>
    </form>
</body>
</html>
```

This page simply uses the CreateUserWizard control and nothing more. This one control enables you to register end users. This particular CreateUserWizard control has a little style applied to it from the Auto Format option found in the control's smart tag, but this control can be as simple as:

```
<asp:CreateUserWizard ID="CreateUserWizard1" Runat="server">
</asp:CreateUserWizard>
```

When this code is run, an end user is presented with the form shown in Figure 8-2.



Figure 8-2

This screen shot shows the form as it would appear when filled out by the end user and includes user information such as the username, password, e-mail, as well as the security question-and-answer section. Clicking the Create User button places this user information into the data store.

The username and password enable the end user to log in to the application through the login server control. A Confirm Password text box is also included in the form to ensure that the password is spelled correctly. An e-mail address is included in case end users forget their login credentials and want the credentials e-mailed to them. Then finally, the security question and answer are used to verify the identity of the end user before any credentials or user information is changed.

After the Create User button is clicked, the end user is presented with a confirmation of the information being stored (see Figure 8-3).

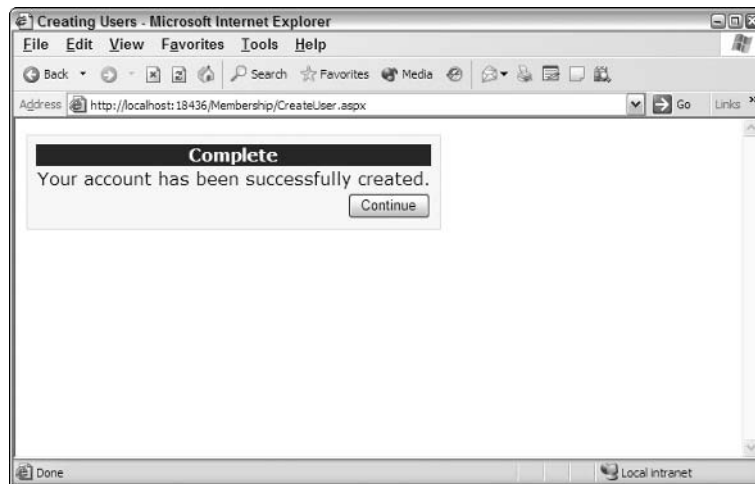


Figure 8-3

Seeing where users are stored

Now that the CreateUserWizard control has been used to add a user to the membership service, take a look at where this information is stored. If you are letting Visual Studio create the Microsoft Access file for you to store the user information, the file was created when the previous example was run. When the example is run, you can click the Refresh button in the Solution Explorer to find the `AspNetDB.mdb` file, which is located in the Data folder. Many different tables are included in this file, but you are only interested in the `aspnet_Membership` table.

When you open the `aspnet_Membership` table, the users you entered are in the system. This is shown in Figure 8-4.

The user password in this table is not stored as clear text, but instead, is hashed. When a user logs into an application that is using the ASP.NET 2.0 membership service, his or her password is immediately hashed and then compared to the hashed password stored in the database. If the two hashed strings do not compare, the passwords are not considered a match.

UserId	Password	PasswordFormat	MobilePIN	Email	PasswordQuestion	PasswordAnswer	IsApproved	CreateDate	LastLoginDate
1	U91vUkwwuGsF	1		ejen@yahoo.co.uk	What is your dog's name	Bucky	<input checked="" type="checkbox"/>	004/13/2004 1:38:14 PM	004/13/2004 1:38:14 PM
0	0	1					<input checked="" type="checkbox"/>	004/13/2004 1:39:01 PM	004/13/2004 1:39:01 PM

Figure 8-4

Working with the CreateUserWizard control

When you work with the CreateUserWizard control, be aware of the ContinueButtonClick and the CreatedUser events. The ContinueButtonClick event is triggered when the Continue button on the second page is clicked after the user has been successfully created (see Listing 8-4).

Listing 8-4: The ContinueButtonClick event

VB

```
Sub CreateUserWizard1_ContinueButtonClick(ByVal sender As Object, _
    ByVal e As System.EventArgs)

    Response.Redirect("Default.aspx")
End Sub
```

C#

```
void CreateUserWizard1_ContinueButtonClick(object sender, EventArgs e)
{
    Response.Redirect("Default.aspx");
}
```

In this example, after the user has been added to the membership service through the form provided by the `CreateUserWizard` control, she can click the Continue button and is redirected to another page in the application. This is done with a simple `Response.Redirect` statement. Remember when you use this event, to add an `OnContinueButtonClick="CreateUserWizard1_ContinueButtonClick"` to the `<asp:CreateUserWizard>` control.

The `CreateUser` event is triggered when a user is successfully created in the data store. The use of this event is shown in Listing 8-5.

Listing 8-5: The `CreateUser` event

VB

```
Sub CreateUserWizard1_CreateUser(ByVal sender As Object, _  
    ByVal e As System.EventArgs)  
  
    ' Code here  
  
End Sub
```

C#

```
void CreateUserWizard1_CreateUser(object sender, EventArgs e)  
{  
    // Code here  
}
```

Use this event if you want to take any additional actions when a user is registered to the service.

Adding users programmatically

You are not limited to using only server controls to register or add new users to the membership service. ASP.NET 2.0 provides a Membership API for performing this task programmatically. This is ideal to create your own mechanics for adding users to the service — or if you are modifying a Web application that was created using ASP.NET 1.0/1.1.

The Membership API includes the `CreateUser` method for adding users to the service. The `CreateUser` method includes three possible signatures:

```
Membership.CreateUser(username As String, password As String)  
  
Membership.CreateUser(username As String, password As String,  
    email As String)  
  
Membership.CreateUser(username As String, password As String,  
    email As String, passwordQuestion As String,  
    passwordAnswer As String, isApproved As Boolean,  
    ByRef status As System.Web.Security.MembershipCreateStatus)
```

You can use this method to create users. The nice thing about this method is that you aren't required to create an instance of the `Membership` class; you use it directly. A simple use of the `CreateUser` method is illustrated in Listing 8-6.

Listing 8-6: Creating users programmatically

VB

```
<%@ Page Language="VB" %>

<script runat="server">
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Try
        Membership.CreateUser(TextBox1.Text, TextBox2.Text)
        Label1.Text = "Successfully created user " & TextBox1.Text
    Catch ex As MembershipCreateUserException
        Label1.Text = "Error: " & ex.ToString()
    End Try
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Creating a User</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Create User</h1>
        <p>Username<br />
            <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        </p>
        <p>Password<br />
            <asp:TextBox ID="TextBox2" Runat="server"
                TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Create User"
                OnClick="Button1_Click" />
        </p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </p>
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            Membership.CreateUser(TextBox1.Text.ToString(),
                TextBox2.Text.ToString());
            Label1.Text = "Successfully created user " + TextBox1.Text;
        }
        catch (MembershipCreateUserException ex)
```

```
        {  
            Label1.Text = "Error: " + ex.ToString();  
        }  
    }  
</script>
```

So, use either the `CreateUserWizard` control or the `CreateUser` method found in the Membership API to create users for your Web applications with relative ease. This functionality was possible in the past with ASP.NET 1.0/1.1, but it was a labor-intensive task. Now with ASP.NET 2.0, you can create users with either a single control or with a single line of code.

Changing how users register with your application

You determine how users register with your applications and what is required of them by the membership provider you choose. Each membership provider has default settings to determine how it behaves established from within the `machine.config` file. If you dig down in the `machine.config` file on your server you find the following code (shown in Listing 8-7).

Listing 8-7: Membership provider settings in the machine.config file

```
<membership defaultProvider="AspNetAccessProvider" userIsOnlineTimeWindow="15" >  
  <providers>  
    <add name="AspNetSqlProvider"  
      type="System.Web.Security.SqlMembershipProvider, System.Web,  
        Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"  
      connectionStringName="LocalSqlServer"  
      enablePasswordRetrieval="false"  
      enablePasswordReset="true"  
      requiresQuestionAndAnswer="false"  
      applicationName="/" "  
      requiresUniqueEmail="false"  
      passwordFormat="Hashed"  
      description="Stores and retrieves membership data from the local  
        Microsoft SQL Server database"  
    />  
  
    <add name="AspNetAccessProvider"  
      type="System.Web.Security.AccessMembershipProvider, System.Web,  
        Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"  
      connectionStringName="AccessFileName"  
      enablePasswordRetrieval="false"  
      enablePasswordReset="true"  
      requiresQuestionAndAnswer="false"  
      applicationName="/" "  
      requiresUniqueEmail="false"  
      passwordFormat="Hashed"  
      description="Stores and retrieves membership data from the local  
        Microsoft Access database file"  
    />  
  </providers>  
</membership>
```

Chapter 8

This section of the `machine.config` file shows the two default membership providers that come with ASP.NET 2.0 — the `AspNetSqlProvider` and the `AspNetAccessProvider`. In the `<membership>` element at the top of the code snippet, you see that the `AspNetAccessProvider` is the default provider and is the second provider detailed.

The important attributes of the `AspNetAccessProvider` definition include the `enablePasswordRetrieval`, `enablePasswordReset`, `requiresQuestionAndAnswer`, `requiresUniqueEmail`, and `PasswordFormat` attributes. The following table defines these attributes.

Attribute	Description
<code>enablePasswordRetrieval</code>	Defines whether the provider supports password retrievals. This attribute takes a <code>Boolean</code> value. The default value is <code>False</code> .
<code>enablePasswordReset</code>	Defines whether the provider supports password resets. This attribute takes a <code>Boolean</code> value. The default value is <code>True</code> . When set to <code>False</code> , passwords cannot be retrieved although they can be changed with a new random password.
<code>requiresQuestionAndAnswer</code>	Specifies whether the provider should require a question and answer combination for when a user is created. This attribute takes a <code>Boolean</code> value, and the default value is <code>False</code> .
<code>requiresUniqueEmail</code>	Defines whether the provider should require a unique e-mail to be specified when the user is created. This attribute takes a <code>Boolean</code> value, and the default value is <code>False</code> . When set to <code>True</code> , only unique e-mail addresses can be entered into the data store.
<code>PasswordFormat</code>	Defines the format in which the password is stored in the data store. The possible values include <code>Hashed</code> , <code>Clear</code> , and <code>Encrypted</code> . The default value is <code>Hashed</code> . Hashed passwords use SHA1, whereas encrypted passwords use Triple-DES encryption.

Asking for credentials

After you have users that can access your Web application using the new membership service provided by ASP.NET 2.0, you can then give these users the means to log into the site. This requires little work on your part. Before you learn the controls that enable users to access your applications, first you should make a few more modifications to the `web.config` file.

Turning off access with the `<authorization>` element

After you make the changes to the `web.config` file by adding the `<authorization>` and `<forms>` elements (Listings 8-1 and 8-2), your Web application is accessible to each and every user that browses to any page your application contains. To prevent open access, you have to deny unauthenticated users access to the pages of your site.

Denying unauthenticated users access to your site is illustrated in Listing 8-8.

Listing 8-8: Denying unauthenticated users

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authentication mode="Forms" />
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

Using the `<authentication>` and `<deny>` elements, you can deny specific users access to your Web application — or (as in this case) simply deny every unauthenticated user (this is what the question mark signifies).

Now that everyone but authenticated users has been denied access to the site, you want to make it easy for viewers of your application to become authenticated users. To do so, use the Login server control.

Using the Login server control

The Login server control enables you to turn unauthenticated users into authenticated users by allowing them to provide login credentials that can be verified in a data store of some kind. In the examples so far, you have used Microsoft Access as the data store.

The first step in using the Login control is to create a new Web page titled `Login.aspx`. This is the default page to which unauthenticated users are redirected in order to obtain their credentials.

The `Login.aspx` page simply needs an `<asp:Login>` control to give the end user everything she needs to become authenticated, as illustrated in Listing 8-9.

Listing 8-9: Providing a login for the end user using the Login control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Login Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:Login ID="Login1" Runat="server">
    </asp:Login>
  </form>
</body>
</html>
```

If the unauthenticated user hits a different page in the application, she is redirected to the `Login.aspx` page. You can see how ASP.NET tracks the location from the URL:

```
http://localhost:18436/Membership/login.aspx?ReturnUrl=%2fMembership%2fDefault.aspx
```

Chapter 8

The login page, using the Login control, is shown in Figure 8-5.

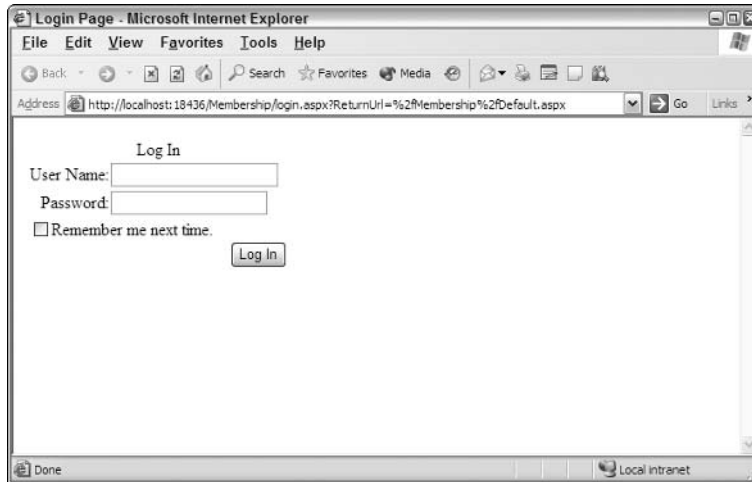


Figure 8-5

From this figure, you can see that the Login control asks the user for a username and password. A check box allows for a cookie to be stored on the client machine. This cookie enables the end user to bypass login in the future. After she is logged in, the end user is returned to the page she originally wanted to access.

You can modify the look and feel of the Login control just as you can for the other controls. One way to do this is by clicking the Auto Format link in the control's smart tag. There you find a list of options for modifying the look and feel of the control (see Figure 8-6).



Figure 8-6

Select the *Elegant* option, for example, and the code is modified. Listing 8-10 shows the code generated for this selection.

Listing 8-10: A formatted Login control

```
<asp:Login ID="Login1" Runat="server" BorderWidth="1px" BorderColor="#CCCC99"
  BorderPadding="0" BorderStyle="Solid" BackColor="#F7F7DE" Font-Names="Verdana"
  Font-Size="10pt">
  <InstructionTextStyle Font-Italic="False"></InstructionTextStyle>
  <TitleTextStyle Font-Bold="True" BackColor="#6B696B"
    ForeColor="#FFFFFF"></TitleTextStyle>
</asp:Login>
```

From this listing, you can see that the `<InstructionTextStyle>` and the `<TitleTextStyle>` subelements are used to modify those particular items displayed by the control. The available styling elements that are included with the Login control include

- ☐ `<CheckboxStyle>`
- ☐ `<FailureTextStyle>`
- ☐ `<HyperLinkStyle>`
- ☐ `<InstructionTextStyle>`
- ☐ `<LabelStyle>`
- ☐ `<SubmitButtonStyle>`
- ☐ `<TextBoxStyle>`
- ☐ `<TitleTextStyle>`

Logging in users programmatically

Besides using the prebuilt mechanics of the Login control, you can also perform this task programmatically using the Membership API. To validate credentials that you receive, you use the `ValidateUser` method. The `ValidateUser` method takes a single signature:

```
ValidateUser(username As String, password As String)
```

This method is illustrated in Listing 8-11.

Listing 8-11: Validating a user's credentials programmatically

VB

```
If Membership.ValidateUser(TextBox1.Text, TextBox2.Text) Then
    FormsAuthentication.RedirectFromLoginPage(TextBox1.Text, False)
Else
    Label1.Text = "You are not registered with the site."
End If
```

C#

```
if (Membership.ValidateUser(TextBox1.Text.ToString(), TextBox2.Text.ToString())) {
    FormsAuthentication.RedirectFromLoginPage(TextBox1.Text.ToString(), false);
}
else {
    Label1.Text = "You are not registered with the site.";
}
```


The `ValidateUser` method returns a `Boolean` value of `True` if the user credentials pass the test and `False` if they do not. From the code snippet in Listing 8-11, you can see that end users whose credentials are verified as correct are redirected from the login page using the `RedirectFromLoginPage` method. This method takes the username and a `Boolean` value that specifies whether the credentials are persisted through a cookie setting.

Working with authenticated users

After users are authenticated, ASP.NET 2.0 provides a number of different server controls and methods that you can use to work with the user details. Included in this collection of tools are the `LoginStatus` and the `LoginName` controls.

The `LoginStatus` server control

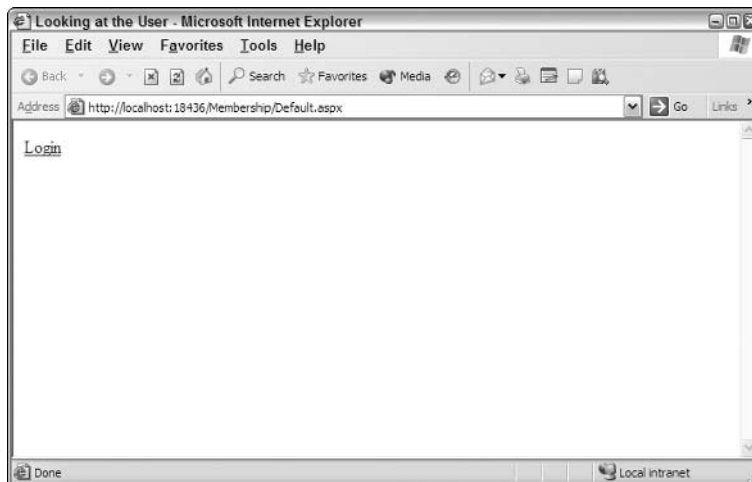
The `LoginStatus` server control enables users to click a link to log in or log out of a site. For a good example of this control, remove the `<deny>` element from the `web.config` file so that the pages of your site are accessible to unauthenticated users. Then code your `Default.aspx` page so that it is similar to the code shown in Listing 8-12.

Listing 8-12: Login and logout features of the `LoginStatus` control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Login or Logout</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:LoginStatus ID="LoginStatus1" Runat="server" />
  </form>
</body>
</html>
```

Running this page gives you a simple page that has only a hyperlink titled `Login`, as shown in Figure 8-7.



Clicking the `Login` hyperlink forwards you to the `Login.aspx` page where you provide your credentials. After the credentials are provided, you are redirected to the `Default.aspx` page — although now the page includes a hyperlink titled `Logout` (see Figure 8-8). The `LinkStatus` control displays one link when the user is unauthenticated and another link when the user is authenticated. Clicking the `Logout` hyperlink logs out the user and redraws the `Default.aspx` page — but with the `Login` hyperlink in place.

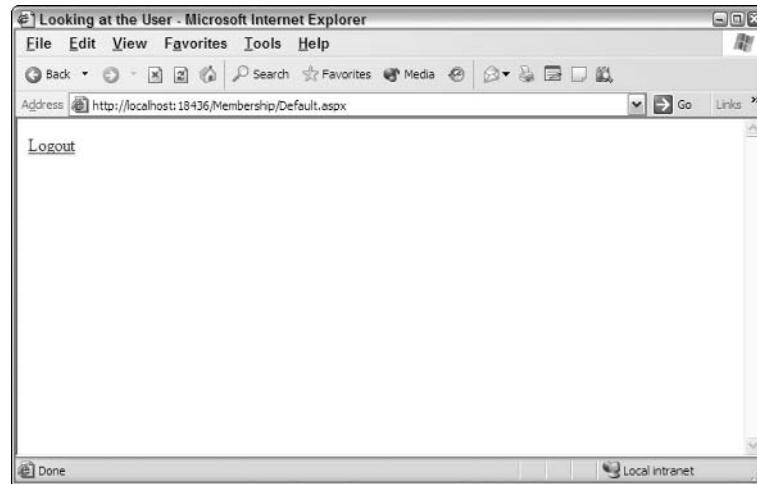


Figure 8-8

The `LoginName` server control

The `LoginName` server control enables you to display the username of the authenticated user. This is a common practice today. For an example of this, change the `Default.aspx` page so that it now includes the authenticated user's login name when that user is logged in, as illustrated in Listing 8-13.

Listing 8-13: Displaying the username of the authenticated user

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:LoginName ID="LoginName1" Runat="server"
            Font-Bold="True" Font-Size="XX-Large" /></p>
    </form>
</body>
</html>
```

When the user logs in to the application and is returned to the `Default.aspx` page, he sees his username displayed, as well as the hyperlink generated by the `LoginStatus` control (see Figure 8-9).

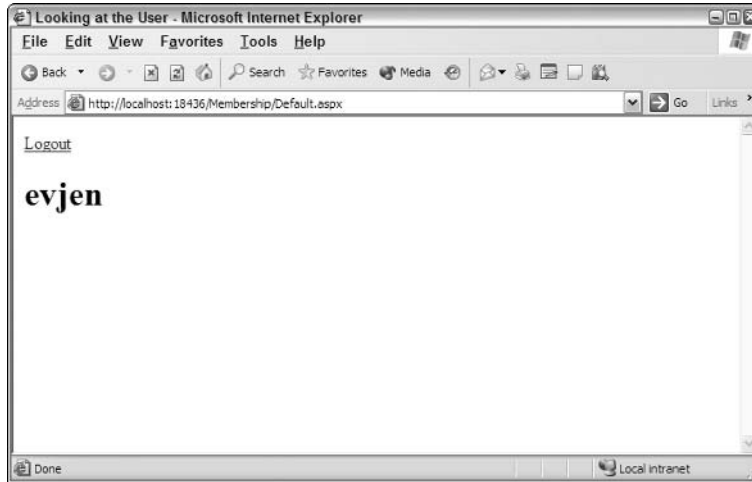


Figure 8-9

Showing the number of users online

One cool feature of the membership service is that you can display how many users are online at a given moment. This is an especially popular option for a portal or a forum that wishes to impress visitors to the site with its popularity.

To show the number of users online, you use the `GetNumberOfUsersOnline` method provided by the `Membership` class. You can add to the `Default.aspx` page shown in Figure 8-9 with the code illustrated in Listing 8-14.

Listing 8-14: Displaying the number of users online

VB

```
<%@ Page Language="VB" %>
```

```
<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = Membership.GetNumberOfUsersOnline.ToString()
    End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Login or Logout</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:LoginName ID="LoginName1" Runat="server"
            Font-Bold="True" Font-Size="XX-Large" /></p>
```

```
<p>There are <asp:Label ID="Label1" Runat="server" Text="0" />
  users online.</p>
</form>
</body>
</html>
```

C#

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

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = Membership.GetNumberOfUsersOnline.ToString();
    }
</script>
```

When the page is generated, it displays the number of users who have logged on in the last 15 minutes. An example of what is generated is shown in Figure 8-10.

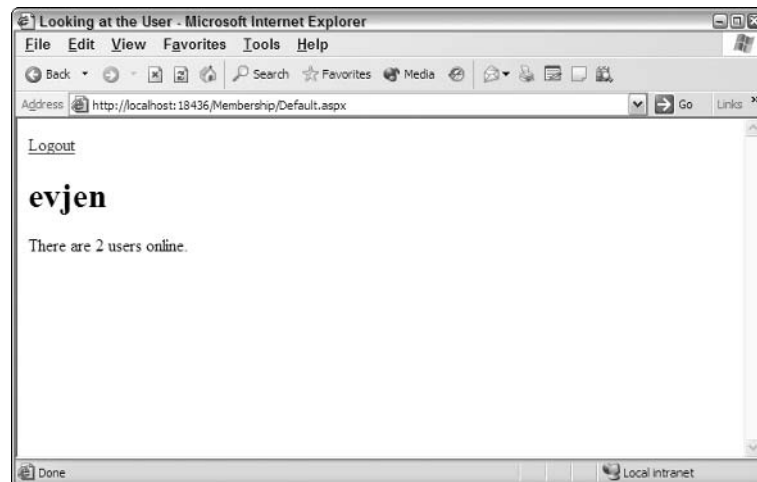


Figure 8-10

You can see that two users have logged on in the last 15 minutes. This 15-minute period is determined in the `machine.config` file from within the `<membership>` element:

```
<membership defaultProvider="AspNetAccessProvider" userIsOnlineTimeWindow="15" >
</membership>
```

By default, the `userIsOnlineTimeWindow` is set to 15. The number is specified here in minutes. To increase the time window, you simply increase this number. In addition to specifying this number from within the `machine.config` file, you can also set this number in the `web.config` file.

Dealing with passwords

Many of us seem to spend our lives online and have username/password combinations for many different Web sites on the Internet. For this reason, end users forget passwords or want to change them every so often. ASP.NET 2.0 provides a couple of new server controls that work with the membership service so that end users can either change their password or retrieve a forgotten password.

The ChangePassword server control

The ChangePassword server control enables end users to change their password directly in the browser. Listing 8-15 shows a use of the ChangePassword control.

Listing 8-15: Allowing users to change their passwords

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Change Your Password</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p><asp:ChangePassword ID="ChangePassword1" Runat="server">
            </asp:ChangePassword><p>
    </form>
</body>
</html>
```

This is a rather simple use of the `<asp:ChangePassword>` control. Running this page produces the results shown in Figure 8-11.

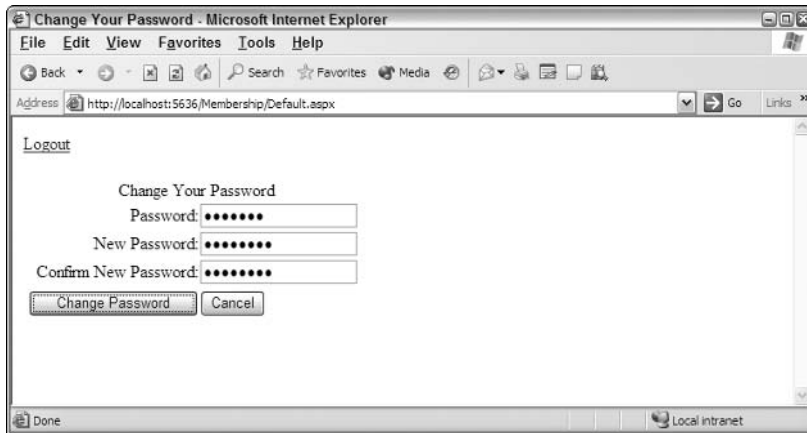


Figure 8-11

The ChangePassword control produces a form that asks for the previous password. It also requires the end user to type the new password twice. Clicking the Change Password button launches an attempt to

change the password if the user is logged in. If the end user isn't logged into the application yet, he or she is redirected to the login page. Only a logged-in user can change his or her password. After the password is changed, the end user is notified (see Figure 8-12).

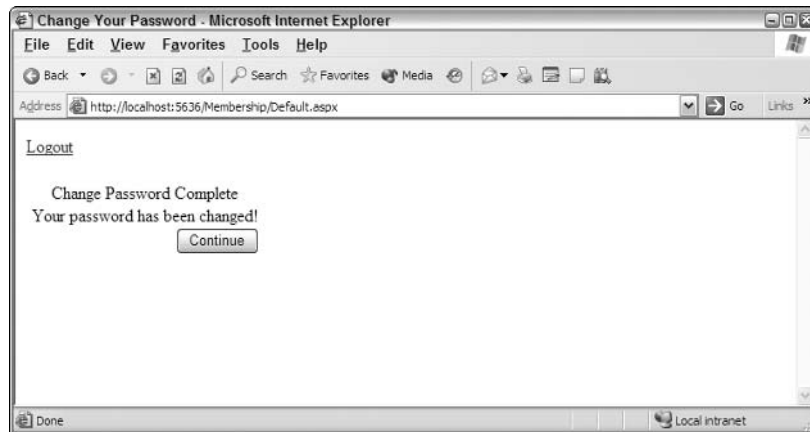


Figure 8-12

Remember that end users are allowed to change their passwords because the `enablePasswordReset` attribute of the membership provider is set to `True`. To deny this capability, set the `enablePasswordReset` attribute to `False`.

The PasswordRecovery server control

People simply forget their passwords. For this reason, you should provide the means to retrieve passwords from your data store. The `PasswordRecovery` server control provides an easy way to accomplish this task.

Password recovery usually means sending the end user's password to him in an e-mail. Therefore, you set up an SMTP server (it might be the same as the application server). You configure for this server in the `web.config` file, as illustrated in Listing 8-16.

Listing 8-16: Setting up the SMTP server in the web.config file

```
<configuration>
  <system.web>

    <smtpMail serverName="localhost" serverPort="25" from="evjen@yahoo.com">
      <fields>
        <add
          name="http://schemas.microsoft.com/cdo/configuration/smtpauthenticate"
          value="0" />
      </fields>
    </smtpMail>

  </system.web>
</configuration>
```

After you have the <smtpMail> element set up correctly, you can start to use the PasswordRecovery control. A simple use of the PasswordRecovery control is shown in Listing 8-17.

Listing 8-17: Using the PasswordRecovery control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Getting Your Password</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:PasswordRecovery ID="PasswordRecovery1" Runat="server">
            <MailDefinition From="evjen@yahoo.com">
                </MailDefinition>
        </asp:PasswordRecovery>
    </form>
</body>
</html>
```

The <asp:PasswordRecovery> element needs a <MailDefinition> subelement. The <MailDefinition> element describes details about the e-mail to be sent to the end user. The minimum requirement is that the From attribute is used, which provides the e-mail address for the From part of the e-mail. The String value of this attribute should be an e-mail address. Other attributes for the <MailDefinition> element include

- ☐ BodyFileName
- ☐ BodyFormat
- ☐ Cc
- ☐ From
- ☐ Priority
- ☐ Subject

When you run this page, the PasswordRecovery control asks for the user's username, as shown in Figure 8-13.

When it has the username, the membership service retrieves the question and answer that was earlier entered by the end user and generates the view shown in Figure 8-14.

If the question is answered correctly, an e-mail containing the password is generated and mailed to the end user. If the question is answered incorrectly, an error message is displayed.

It is important to change some of your membership service settings in order for this entire process to work. At present, it won't work because of the way in which a user's password is hashed. The membership service data store isn't storing the actual password — just this hashed version of it. Of course, it is useless for an end user to receive a hashed password.

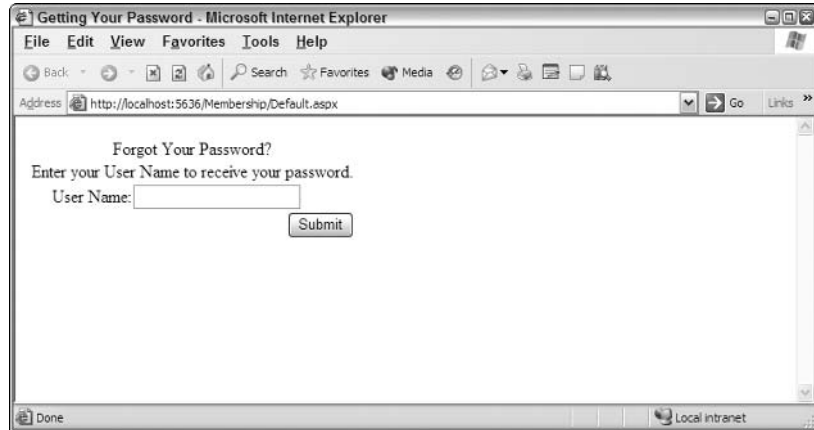


Figure 8-13



Figure 8-14

In order for you to be able to send back an actual password to the user, you must change how the passwords are stored in the membership service data store. This is done (as stated earlier in the chapter) by changing `PasswordFormat` attribute of your membership data provider. The other possible values (besides `Hashed`) are `Clear` and `Encrypted`. Changing it to either `Clear` or `Encrypted` makes it possible for the passwords to be sent back to the end user in a readable format.

ASP.NET 2.0 Authorization

Now that you can deal with the registration and authentication of users who want to access your Web applications, the next step is authorization. What are they allowed to see and what roles do they take? These are important questions for any Web application. First, learn how to show only certain items to authenticated users while showing different items to unauthenticated users.

Using the LoginView server control

The LoginView server control allows you to control who views what information on a particular part of a page. Using the LoginView control, you can dictate which parts of the pages are for authenticated users and which parts of the pages are for unauthenticated users. Listing 8-18 shows an example of this control.

Listing 8-18: Controlling information viewed via the LoginView control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Changing the View</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:LoginStatus ID="LoginStatus1" Runat="server" />
        <p>
            <asp:LoginView ID="LoginView1" Runat="server">
                <LoggedInTemplate>
                    Here is some REALLY important information that you should know
                    about all those people that are not authenticated!
                </LoggedInTemplate>
                <AnonymousTemplate>
                    Here is some basic information for you.
                </AnonymousTemplate>
            </asp:LoginView>
        <p>
    </form>
</body>
</html>
```

The `<asp:LoginView>` control is a templated control that takes two possible subelements — the `<LoggedInTemplate>` and `<AnonymousTemplate>` elements. In this case, the information defined in the `<AnonymousTemplate>` section (see Figure 8-15) is for unauthenticated users and is quite different from what authenticated users see defined in the `<LoggedInTemplate>` section (see Figure 8-16).

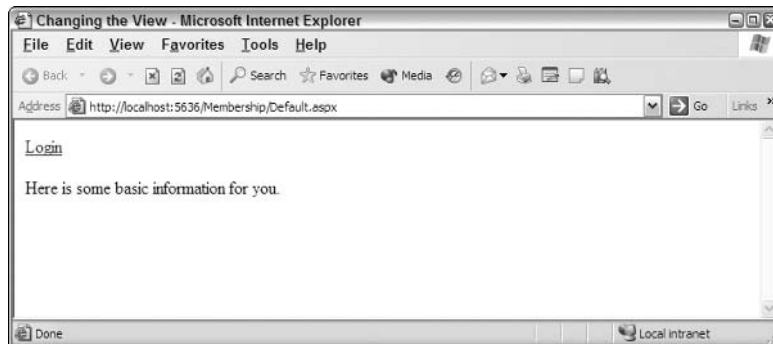


Figure 8-15

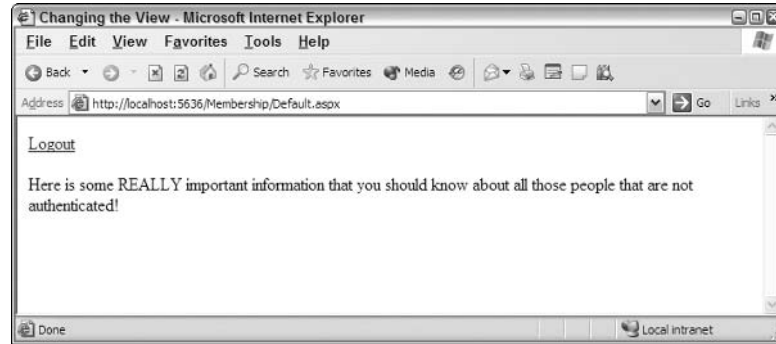


Figure 8-16

Only simple ASCII text is placed inside both of these templates, but you can actually place anything else within the template including additional server controls. This means that you can show entire sections of pages, including forms, from within the templated sections.

Setting up your Web site for role management

In addition to the membership service just reviewed, ASP.NET 2.0 provides you with the other side of the end user management service — the ASP.NET role management service. The membership service covered all the details of authentication for your applications, whereas the role management service covers authorization. Just as the membership service can use any of the data providers listed earlier, the role management service can use the same providers plus a couple of others. In fact, this service is comparable to the membership service in many ways. Figure 8-17 shows you a simple diagram that details some the particulars of the role management service.

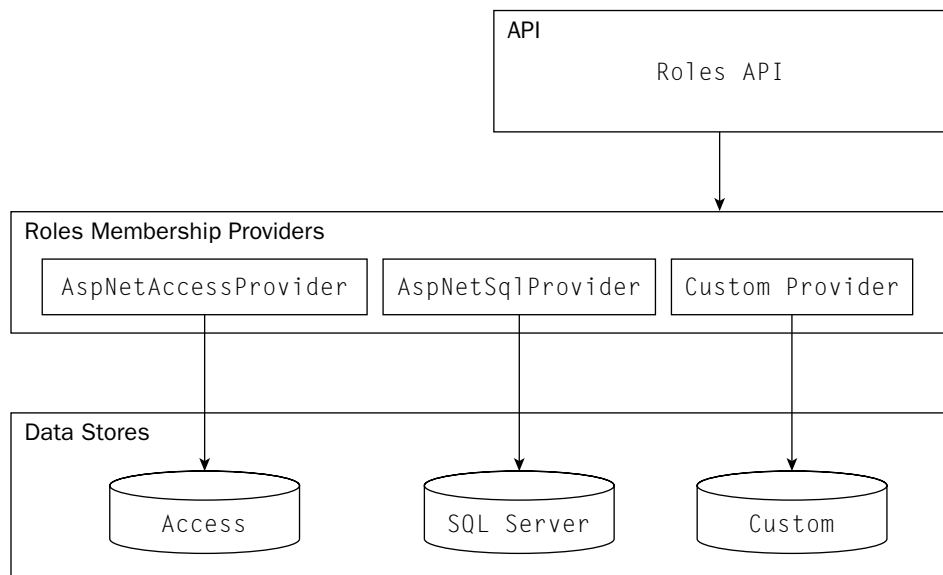


Figure 8-17

Making changes to the <roleManager> section

The first step in working with the role management service is to change any of the role management provider behaviors either in the `machine.config` or from the `web.config` files. If you look in the `machine.config` file, you see an entire section that deals with the role management service (see Listing 8-19).

Listing 8-19: Role management provider settings in the `machine.config` file

```
<roleManager
  enabled="false" cacheRolesInCookie="false" cookieName=".ASPXROLES"
  cookieTimeout="30" cookiePath="/" cookieRequireSSL="false"
  cookieSlidingExpiration="true" createPersistentCookie="false"
  cookieProtection="All" defaultProvider="AspNetAccessProvider" >
  <providers>
    <add name="AspNetSqlProvider" type="System.Web.Security.SqlRoleProvider,
      System.Web, Version=2.0.3600.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LocalSqlServer"
      applicationName="/"
      description="Stores and retrieves roles data from the local
        Microsoft SQL Server database" />

    <add name="WindowsToken"
      type="System.Web.Security.WindowsTokenRoleProvider, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      description="Retrieves roles data from the Windows authenticated token
        for the request" />

    <add name="AspNetAccessProvider"
      type="System.Web.Security.AccessRoleProvider, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="AccessFileName"
      applicationName="/"
      description="Stores and retrieves roles data from the local
        Microsoft Access database file" />

    <add name="AspNetAzManProvider"
      type="System.Web.Security.AzManRoleProvider, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="AzManStore"
      applicationName="MyApplication"
      scopeName="MyScope"
      description="Stores and retrieves roles data from the
        Authorization Manager application store" />
  </providers>
</roleManager>
```

The role management service defines its settings from within the `machine.config` file, as shown in the previous code listing. You can make changes to these settings either directly in the `machine.config` file or by overriding these settings in the `web.config` file (thereby making changes only to the application at hand).

The main settings are defined in the `<roleManager>` element. Some of the attributes of the `<roleManager>` element are defined in the following table.

Attribute	Description
<code>enabled</code>	Defines whether the role management service is enabled for the application. This attribute takes a <code>Boolean</code> value and is set to <code>False</code> by default. This means that the role management service is disabled by default. Therefore, you must change this value to <code>True</code> in either the <code>machine.config</code> or the <code>web.config</code> file first. This value is set to <code>False</code> by default to avoid breaking changes that would occur for people migrating from ASP.NET 1.0/1.1 to ASP.NET 2.0.
<code>cacheRolesInCookie</code>	Defines whether the roles of the user can be stored within a cookie on the client machine. This attribute takes a <code>Boolean</code> value and is set to <code>True</code> by default. This is an ideal situation because retrieving the roles from the cookie prevents ASP.NET from looking up the roles of the user via the role management provider. Set to <code>False</code> if you want the roles to be retrieved via the provider for all instances.
<code>cookieName</code>	Defines the name used for the cookie sent to the end user for role management information storage. By default, this cookie is named <code>.ASPXROLES</code> , and you probably won't change this.
<code>cookieTimeout</code>	Defines the amount of time (in minutes) after which the cookie expires. The default value is 30 minutes.
<code>cookieRequireSSL</code>	Defines whether you are going to want to require that the role management information be sent over an encrypted wire (SSL) instead of sending credentials as clear text. The default value is <code>False</code> .
<code>cookieSlidingExpiration</code>	Specifies whether the timeout of the cookie is on a sliding scale. The default value is <code>True</code> . This means that the end user's cookie does not expire until 30 minutes (or the time specified in the <code>cookieTimeout</code> attribute) after the last request to the application has been made. If the value of the <code>cookieSlidingExpiration</code> attribute is set to <code>False</code> , the cookie expires 30 minutes from the first request.
<code>createPersistentCookie</code>	Specifies whether the cookie that is created should never expire, but instead remain alive indefinitely. The default setting is <code>False</code> because this is not always advisable for security reasons.
<code>cookieProtection</code>	Specifies the amount of protection that you want to apply to the cookie that is stored on the end user's machine for management information. The possible settings include <code>All</code> , <code>None</code> , <code>Encryption</code> , and <code>Validation</code> . You should always attempt to use <code>All</code> .
<code>defaultProvider</code>	Defines the provider used for the role management service. By default, it is set to <code>AspNetAccessProvider</code> .

Making changes to the web.config file

The next step is to configure your `web.config` file so that it can work with the role management service. Certain pages or subsections of your application may be accessible only to people with specific roles. To manage this access, you define the access rights in the `web.config` file. The necessary changes are shown in Listing 8-20.

Listing 8-20: Changing the web.config file

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <system.web>
    <roleManager enabled="true"/>
    <authentication mode="Forms" />
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>

  <location path="AdminPage.aspx">
    <system.web>
      <authorization>
        <allow roles="AdminPageRights" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>

</configuration>
```

This `web.config` file is doing a couple of things. First, the function of the first `<system.web>` section is no different from that of the membership service shown earlier in the chapter. The `<deny>` element is denying all unauthenticated users across the board.

The second section of this `web.config` file is rather interesting. The `<location>` element is used to define the access rights of a particular page in the application (`AdminPage.aspx`). In this case, only users contained in the `AdminPageRights` role are allowed to view the page, whereas all other users — regardless whether they are authenticated — are not allowed to view the page. When using the asterisk (*) as a value of the `users` attribute of the `<deny>` element, you are saying that all users (regardless of whether they are authenticated) are not allowed to access the resource being defined. This overriding denial of access, however, is broken open a bit via the use of the `<allow>` element, which allows users contained within a specific role.

Adding and retrieving application roles

Now that the `machine.config` and the `web.config` files are in place, you can add roles to the role management service. The role management service, just like the membership service, uses data stores to store information about the users. In these examples, I focus primarily on using Microsoft Access as the provider because it is the default provider.

To get Microsoft Access ready for use, read the instructions outlined earlier in the section “Adding users.”

One big difference between the role management service and the membership service is that no server controls are used for the role management service. You manage the application’s roles and the user’s role details through a new Roles API or through the Web Site Administration Tool provided with ASP.NET 2.0. Listing 8-21 shows how to use some of the new methods to add roles to the service.

Listing 8-21: Adding roles to the application**VB**

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        ListBoxDataBind()
    End Sub

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Roles.CreateRole(TextBox1.Text)
        ListBoxDataBind()
    End Sub

    Sub ListBoxDataBind()
        ListBox1.DataSource = Roles.GetAllRoles()
        ListBox1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>Role Manager</h1>
        Add Role:<br />
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <p><asp:Button ID="Button1" Runat="server" Text="Add Role to Application"
            OnClick="Button1_Click" /></p>
        Roles Defined:<br />
        <asp:ListBox ID="ListBox1" Runat="server">
        </asp:ListBox>
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
```

(continued)

Listing 8-21: (continued)

```
        ListBoxDataBind();
    }

    void Button1_Click(object sender, EventArgs e)
    {
        Roles.CreateRole(TextBox1.Text.ToString());
        ListBoxDataBind();
    }

    void ListBoxDataBind()
    {
        ListBox1.DataSource = Roles.GetAllRoles();
        ListBox1.DataBind();
    }
</script>
```

This example enables you to enter roles into the text box and then submit them to the role management service. The roles contained in the role management service are then displayed in the list box, as illustrated in Figure 8-18.



Figure 8-18

To enter the roles into the management service, you simply use the `CreateRole` method of the `Roles` class. Just as with the `Membership` class, you don't instantiate the `Roles` class. To add roles to the role management service, use the `CreateRole` method that takes only a single parameter — the name of the role as a `String` value:

```
Roles.CreateRole(rolename As String)
```

With this method, you can create as many roles as you want, but each role must be unique — otherwise an exception is thrown.

To retrieve the roles that are in the application's role management service (such as the list of roles displayed in the list box from the earlier example), you use the `GetAllRoles` method of the `Roles` class. This method returns a `String` collection of all the available roles in the service:

```
Roles.GetAllRoles()
```

Deleting roles

It would be just great to sit and add roles to the service all day long. Every now and then, however, you must delete roles from the service as well. Deleting roles is just as easy as adding roles to the role management service. To delete a role, you use one of the `DeleteRole` method signatures. The first option of the `DeleteRole` method takes a single parameter — the name of the role as a `String` value. The second option takes the name of the role plus a `Boolean` value that determines whether to throw an exception when one or more members are contained within that particular role:

```
Roles.DeleteRole(rolename As String)
```

```
Roles.DeleteRole(rolename As String, throwOnPopulatedRole As Boolean)
```

Listing 8-22 is a partial code example that builds on Listing 8-21. For this example, add an additional button, which initiates a second button-click event that deletes the role from the service.

Listing 8-22: Deleting roles from the application

VB

```
Sub DeleteButton_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    For Each li As ListItem In ListBox1.Items
        If li.Selected = True Then
            Roles.DeleteRole(li.ToString())
        End If
    Next
    ListBoxDataBind()
End Sub
```

C#

```
void DeleteButton_Click(object sender, EventArgs e)
{
    foreach (ListItem li in ListBox1.Items) {
        if (li.Selected == true) {
            Roles.DeleteRole(li.ToString());
        }
    }
    ListBoxDataBind();
}
```

This example deletes the selected items from the `ListBox` control. If more than one selection is made (meaning that you have placed the attribute `SelectionMode="Multiple"` in the `ListBox` control), each of the roles is deleted from the service, in turn, in the `For Each` loop. Although `Roles.DeleteRole(li.ToString())` is used to delete the role, `Roles.DeleteRole(li.ToString(), True)` could also be used to make sure that no roles are deleted if that role contains any members.

Adding users to roles

Now that the roles are in place and it is possible to also delete them if required, the next step is adding users to the roles created. A role doesn't do much good if no users are associated with the role. To add a single user to a single role, you use the following construct:

```
Roles.AddUserToRole(username As String, rolename As String)
```

To add a single user to multiple roles at the same time, you use this construct:

```
Roles.AddUserToRoles(username As String, rolenames() As String)
```

To add multiple users to a single role, you use the following construct:

```
Roles.AddUsersToRole(usernames() As String, rolename As String)
```

Then finally, to add multiple users to multiple roles, you use the following construct:

```
Roles.AddUsersToRoles(usernames() As String, rolenames() As String)
```

The parameters that can take collections, whether they are `usernames()` or `rolenames()`, are presented to the method as `String` arrays.

Getting all the users of a particular role

Looking up information is easy in the role management service, whether you are determining which users are contained within a particular role, or whether you want to know the roles that a particular user belongs to.

Methods are available for either of these scenarios. First, look at how to determine all the users contained in a particular role, as illustrated in Listing 8-23.

Listing 8-23: Looking up users in a particular role

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        DropDownList1.DataBind()
    End Sub

    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        GridView1.DataSource = Roles.GetUsersInRole(DropDownList1.SelectedValue)
        GridView1.DataBind()
        DropDownList1.DataBind()
    End Sub

    Sub DropDownList1_DataBind()
        DropDownList1.DataSource = Roles.GetAllRoles()
        DropDownList1.DataBind()
    End Sub
End Sub
```

```

        End Sub

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        Roles:
        <asp:DropDownList ID="DropDownList1" Runat="server">
        </asp:DropDownList>
        <asp:Button ID="Button1" Runat="server" Text="Get Users In Role"
            OnClick="Button1_Click" />
        <br />
        <br />
        <asp:GridView ID="GridView1" Runat="server">
        </asp:GridView>
    </form>
</body>
</html>

```

C#

```

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

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        DropDownDataBind();
    }

    void Button1_Click(object sender, EventArgs e)
    {
        GridView1.DataSource = Roles.GetUsersInRole(DropDownList1.SelectedValue);
        GridView1.DataBind();
        DropDownDataBind();
    }

    void DropDownDataBind()
    {
        DropDownList1.DataSource = Roles.GetAllRoles();
        DropDownList1.DataBind();
    }
</script>

```

This page creates a drop-down list that contains all the roles for the application. Clicking the button displays all the users for the selected role. Users of a particular role are determined using the `GetUsersInRole` method. This method takes a single parameter — a `String` value representing the name of the role:

```
Roles.GetUsersInRole(rolename As String)
```

When run, the page looks similar to the page shown in Figure 8-19.

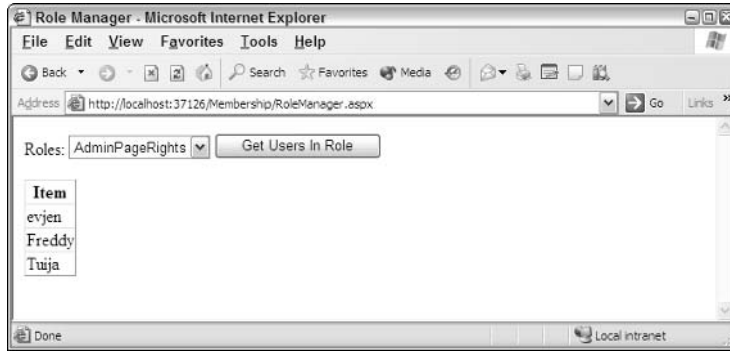


Figure 8-19

Getting all the roles of a particular user

To determine all the roles for a particular user, create a page with a single text box and a button. In the text box, you type the name of the user, and a button click initiates the retrieval and populates a GridView control. The button click event (where all the action is) is illustrated in Listing 8-24.

Listing 8-24: Getting all the roles of a specific user

VB

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    GridView1.DataSource = Roles.GetRolesForUser(TextBox1.Text)
    GridView1.DataBind()
End Sub
```

C#

```
void Button1_Click(object sender, EventArgs e)
{
    GridView1.DataSource = Roles.GetRolesForUser(TextBox1.Text.ToString());
    GridView1.DataBind();
}
```

The preceding code produces something similar to what is shown in Figure 8-20.

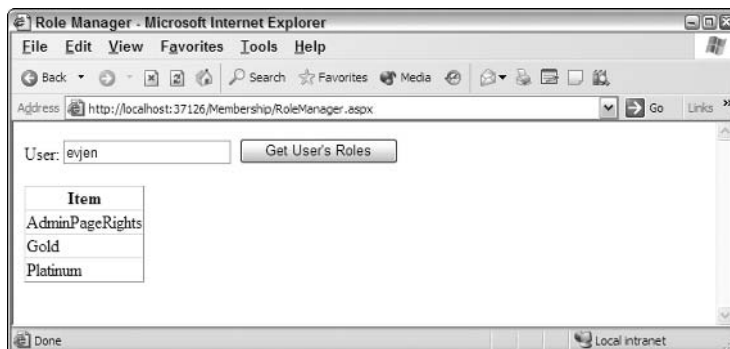


Figure 8-20

To get the roles of a particular user, you simply use the `GetRolesForUser` method. This method has two possible signatures. The first is shown in the preceding example — a `String` value that represents the name of the user. The other option is an invocation of the method without any parameters listed. This returns the roles of the user who has logged into the membership service.

Removing users from roles

In addition to adding users to roles, you can also easily remove users from roles. To delete or remove a single user from a single role, you use the following construct:

```
Roles.RemoveUserFromRole(username As String, rolename As String)
```

To remove a single user from multiple roles at the same time, you use this construct:

```
Roles.RemoveUserFromRoles(username As String, rolenames() As String)
```

To remove multiple users from a single role, you use the following construct:

```
Roles.RemoveUsersFromRole(usernames() As String, rolename As String)
```

Then finally, to remove multiple users from multiple roles, you use the following construct:

```
Roles.RemoveUsersFromRoles(usernames() As String, rolenames() As String)
```

The parameters shown as collections, whether they are `usernames()` or `rolenames()`, are presented to the method as `String` arrays.

Checking users in roles

One final action you can take is checking whether or not a particular user is in a role. You can go about this in couple of ways. The first is using the `IsUserInRole` method.

The `IsUserInRole` method takes two parameters — the username and the name of the role:

```
Roles.IsUserInRole(username As String, rolename As String)
```

This method returns a `Boolean` value on the status of the user, and it can be used as shown in Listing 8-25.

Listing 8-25: Checking a user's role status

VB

```
If (Roles.IsUserInRole(TextBox1.Text, "AdminPageRights")) Then  
    ' perform action here  
End If
```

C#

```
If (Roles.IsUserInRole(TextBox1.Text.ToString(), "AdminPageRights"))  
{  
    // perform action here  
}
```

Chapter 8

The other option, in addition to the `IsUserInRole` method, is to use `FindUsersInRole`. This method enables you make a name search against all the users in a particular role. The `FindUsersInRole` method takes two parameters — the name of the role and the username, both as `String` values:

```
Roles.FindUsersInRole(rolename As String, username As String)
```

Listing 8-26 shows an example of this method.

Listing 8-26: Checking for a specific user in a particular role

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        GridView1.DataSource = _
            Roles.FindUsersInRole("AdminPageRights", TextBox1.Text)
        GridView1.DataBind()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Role Manager</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
        <asp:Button ID="Button1" Runat="server" Text="Button"
            OnClick="Button1_Click" />
        <p><asp:GridView ID="GridView1" Runat="server">
            </asp:GridView></p>
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Button1_Click(object sender, EventArgs e)
    {
        GridView1.DataSource =
            Roles.FindUsersInRole("AdminPageRights", TextBox1.Text.ToString());
        GridView1.DataBind();
    }
</script>
```

Using the Web Site Administration Tool

Many of the actions shown in this chapter can also be performed through the Web Site Administration Tool shown in Figure 8-21.

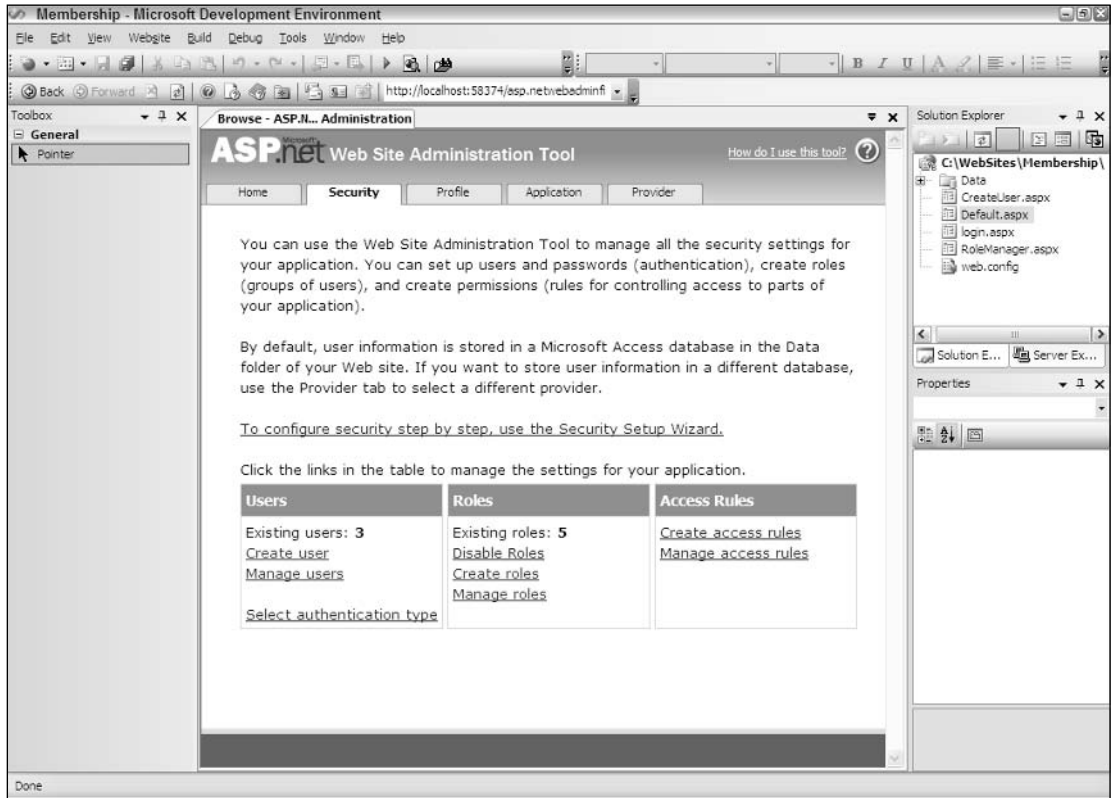


Figure 8-21

Although you can easily use this tool to perform all the actions for you, often you perform these actions through your own applications as well. It is important to know all the possibilities when programming an ASP.NET application.

The Web Site Administration Tool is detailed in Chapter 14.

Summary

This chapter covered two outstanding new additions to ASP.NET 2.0. These are probably my favorite new ASP.NET features. The membership and role management services that are now a part of ASP.NET make managing users and their roles almost trivial.

This chapter reviewed both the Membership and Roles APIs and the controls that also utilize these APIs. These new controls and APIs follow the same data provider models as the rest of ASP.NET 2.0. The examples were presented using Microsoft Access, but you can also use Microsoft SQL Server for the back-end storage.

Chapter 9, “Personalization,” builds on this chapter and shows you how to use SQL Server as the back-end data store as well. The lessons you learn in the next chapter can also be applied here.

9

Personalization

Many Web applications must be customized to contain information that is specific to the end user who is presently viewing the page. In the past, the developer usually provided storage of personalization properties for end users viewing the page by means of cookies, the `Session` object, or the `Application` object. Cookies enabled storage of persistent items so that when the end user returned to a Web page, any settings related to him were retrieved. Cookies aren't the best way to approach persistent user data storage, however, because they are not accepted by all computers, and also because a crafty end user can easily alter them.

As you saw in the previous chapter, ASP.NET 2.0's membership and role management capabilities are ways for ASP.NET to conveniently store information about the user. How can you, as the developer, use the same mechanics to store custom information?

ASP.NET 2.0 provides you with a new and outstanding feature — *Personalization*. The ASP.NET Personalization engine provided with this latest release can make an automatic association between the end user viewing the page and any data points stored for that user. The personalization properties that are maintained on a per-user basis are stored on the server and not on the client. These items are conveniently placed in a data store of your choice (such as Microsoft's SQL Server) and, therefore, the end user can access these personalization properties on later site visits.

This new feature is an ideal way to start creating highly customizable and user-specific sites without building any of the plumbing beforehand. The new Personalization feature is yet another way that the ASP.NET team is making the lives of developers easier by making them more productive.

The Personalization Model

The Personalization model provided with ASP.NET 2.0 is simple and, as with most items that come with ASP.NET, it is an extensible model as well. Figure 9-1 shows a simple diagram that outlines the new Personalization model.

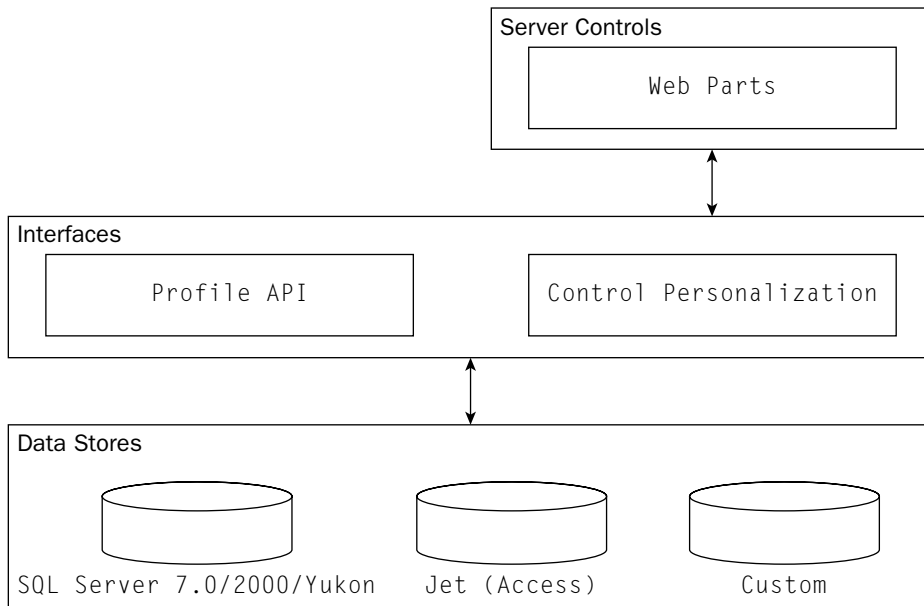


Figure 9-1

From this diagram, you can see the three layers in this model. First, look at the middle layer of the Personalization model — the Personalization Services layer. This layer contains the Profile API. This new Profile API layer enables you to program your end user's data points into one of the lower-layer data stores. Also included in this layer are the server control personalization capabilities, which are important for the Portal Framework and the use of Web Parts. The Portal Framework and Web Parts are discussed in Chapter 10.

Although controls are built into ASP.NET that utilize the new personalization capabilities for storing information about the page settings, you can also use this new engine to store your own data points. Just like Web Parts, these points can be used within your ASP.NET pages.

Below the Personalization Services layer, you find the two default personalization data providers — one for working with Microsoft's SQL Server, and another for working with Microsoft Access Jet data stores. You are not limited to just these two data stores when applying the new personalization features of ASP.NET 2.0; you can also extend the model and create a custom data provider for the personalization engine.

Now that you have looked briefly at the personalization model, you can begin using it by creating some stored personalization properties that can be used later within your applications.

Creating Personalization Properties

The nice thing about creating custom personalization properties is that you can do so easily, and you gain a strongly typed access to the items you create. It is also possible to create personalization properties that are used only by authenticated users, and also some that anonymous users can utilize. These

data points are powerful — mainly because you can start using them immediately in your application. First, try creating some simple personalization properties. Later, you learn how to use these personalization properties within your application.

Adding a simple personalization property

The first step is to decide what data items you are going to store from the user. For our example, create a few items about the user that you might want to use within your application; assume that you want to store the following information about the user:

- ☐ First name
- ☐ Last name
- ☐ Last visited
- ☐ Age
- ☐ Member

ASP.NET has a heavy dependency on storing configurations inside XML files, and the ASP.NET 2.0 personalization engine is no different. All these customization points concerning the end user are defined and stored within the `web.config` file of your application. This is illustrated in Listing 9-1.

Listing 9-1: Creating personalization properties in the `web.config` file

```
<configuration>
  <system.web>

    <profile inherits="System.Web.Profile.HttpProfileBase, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">

      <properties>

        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" />

      </properties>

    </profile>

  </system.web>
</configuration>
```

Within the `web.config` file and nested within the `<system.web>` section of the file, you create a `<profile>` section in order to work with the ASP.NET 2.0 Personalization engine. Within this `<profile>` section of the `web.config` file, you need a `<properties>` section. In this section, you can define all the properties you want the personalization engine to store.

From this code example, you can see that it was rather easy to define simple properties using the `<add>` element. This element simply takes the `name` attribute, which takes the name of the property you want to persist.

It's just as easy to use these personalization properties as it is to define them. The next section looks at how to use these definitions in an application.

Using personalization properties

Now that you have defined the personalization properties in the `web.config` file, it's possible to use these items in code. For an example, I create a simple form that asks for some of this information from the end user. On the `Button_Click` event, the data is stored in the personalization engine. Listing 9-2 shows an example of this.

Listing 9-2: Using the defined personalization properties

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Profile.FirstName = TextBox1.Text
        Profile.LastName = TextBox2.Text
        Profile.Age = TextBox3.Text
        Profile.Member = Radiobuttonlist1.SelectedItem.Text
        Profile.LastVisited = DateTime.Now().ToString()

        Label1.Text = "Stored information includes:<p>" & _
            "First name: " & Profile.FirstName & _
            "<br>Last name: " & Profile.LastName & _
            "<br>Age: " & Profile.Age & _
            "<br>Member: " & Profile.Member & _
            "<br>Last visited: " & Profile.LastVisited
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Storing Personalization</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>First Name:
        <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox></p>
        <p>Last Name:
        <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox></p>
        <p>Age:
        <asp:TextBox ID="TextBox3" Runat="server" Width="50px"
            MaxLength="3"></asp:TextBox></p>
        <p>Are you a member?
```

```

        <asp:RadioButtonList ID="Radiobuttonlist1" Runat="server">
            <asp:ListItem Value="1">Yes</asp:ListItem>
            <asp:ListItem Value="0" Selected="True">No</asp:ListItem>
        </asp:RadioButtonList></p>
        <p><asp:Button ID="Button1" Runat="server" Text="Submit"
            OnClick="Button1_Click" />
        </p>
        <hr /><p>
        <asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>

```

C#

```

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

<script runat="server">
    void Button1_Click(object sender, EventArgs e)
    {
        Profile.FirstName = TextBox1.Text;
        Profile.LastName = TextBox2.Text;
        Profile.Age = TextBox3.Text;
        Profile.Member = Radiobuttonlist1.SelectedItem.Text;
        Profile.LastVisited = DateTime.Now().ToString();

        Label1.Text = "Stored information includes:<p>" +
            "First name: " + Profile.FirstName +
            "<br>Last name: " + Profile.LastName +
            "<br>Age: " + Profile.Age +
            "<br>Member: " + Profile.Member +
            "<br>Last visited: " + Profile.LastVisited;
    }
</script>

```

This is similar to the way you worked with the `Session` object in the past, but note that the personalization properties you are storing and retrieving are not key based. Therefore, when working with them you don't need to remember key names.

By default, these items are stored as type `String`, and you have early-bound access to the items stored. To store an item, you simply populate the personalization property directly using the `Profile` object:

```
Profile.FirstName = TextBox1.Text
```

To retrieve the same information, you simply grab the appropriate property of the `Profile` class as shown here:

```
Label1.Text = Profile.FirstName
```

The great thing about using the `Profile` class and all the personalization properties defined in code is that this method provides `IntelliSense`. When working with the `Profile` class, all the items you define are listed as available options, as illustrated in Figure 9-2.

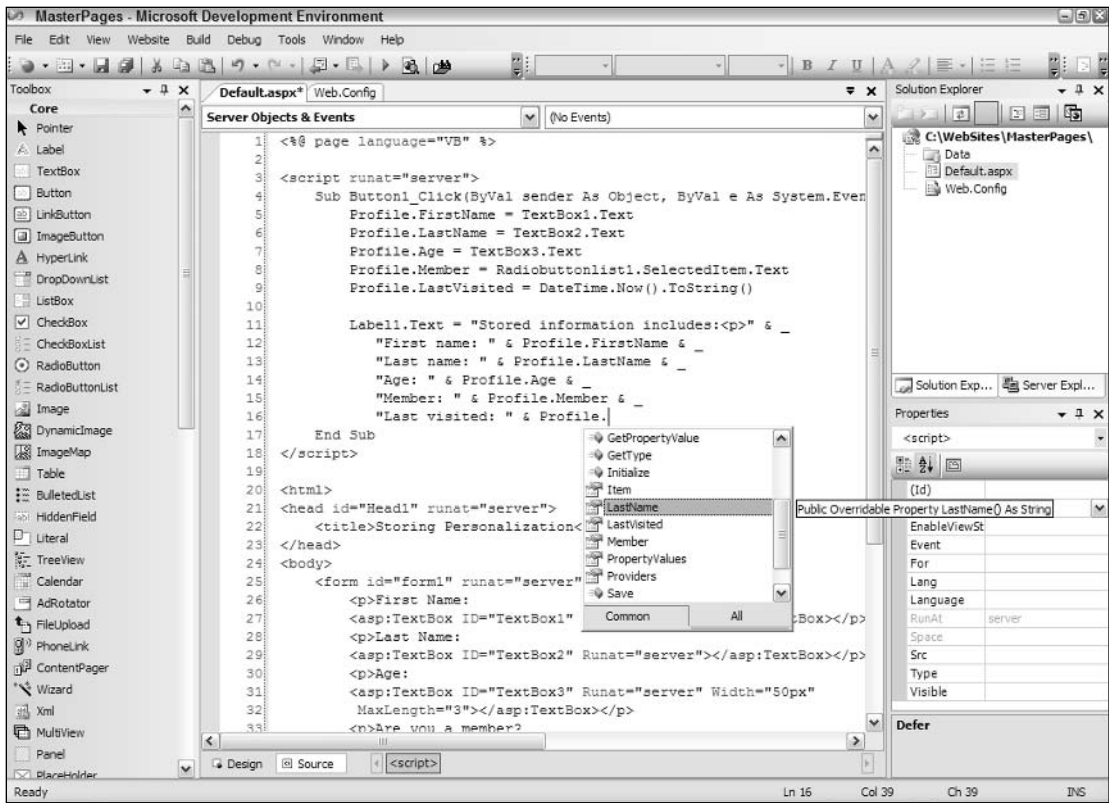


Figure 9-2

All these properties are accessible in IntelliSense because the `Profile` class is hidden and dynamically compiled behind the scenes whenever you save the personalization changes made to the `web.config` file. After these items are saved in the `web.config` file, these properties are available to you throughout your application.

When run, the page from Listing 9-2 produces the results shown in Figure 9-3.

Figure 9-3

In addition to using early-bound access techniques, you can also use late-bound access for the items that you store in the personalization engine. This technique is illustrated in Listing 9-3.

Listing 9-3: Using late-bound access

VB

```
Dim myFirstName As String

myFirstName = Profile.PropertyValues("FirstName").PropertyValue.ToString()
```

C#

```
string myFirstName;

myFirstName = (string) Profile.PropertyValues["FirstName"].PropertyValue;
```

Whether it is early-bound access or late-bound access, you can easily store and retrieve personalization properties for a particular user using this new capability afforded by ASP.NET 2.0. All this is done in the personalization engine's simplest form — now take a look at how you can customize for specific needs in your applications.

Adding a group of personalization properties

If you want to store a large number of personalization properties about a particular user, remember, you are not just storing personalization properties for a particular page, but for the *entire* application. This means that items you have stored about a particular end user somewhere in the beginning of the application can be retrieved later for use on any other page within the application. Because different sections of your Web applications store different personalization properties, you sometimes end up with a large collection of items to be stored and then made accessible.

To make it easier not only to store the items, but also to retrieve them, the personalization engine enables you to store your personalization properties in groups. This is illustrated in Listing 9-4.

Listing 9-4: Creating personalization groups in the web.config file

```
<configuration>
  <system.web>

    <profile inherits="System.Web.Profile.HttpProfileBase, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">

      <properties>

        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />

        <group name="MemberDetails">
          <add name="Member" />
          <add name="DateJoined" />
          <add name="PaidDuesStatus" />
          <add name="Location" />
        </group>

        <group name="FamilyDetails">
          <add name="MarriedStatus" />
          <add name="DateMarried" />
          <add name="NumberChildren" />
          <add name="Location" />
        </group>

      </properties>

    </profile>

  </system.web>
</configuration>
```

From the code in Listing 9-4, which is placed within the `web.config` file, you can see that two groups are listed. The first group is the `MemberDetails` group, which has four specific items defined; the second group — `FamilyDetails` — has three other related items defined. Personalization groups are

defined using the `<group>` element within the `<properties>` definition. The name of the group is specified using the `name` attribute, just as you specify the `<add>` element. You can have as many groups defined as you deem necessary or as have been recommended as good practice to employ.

Using grouped personalization properties

From Listing 9-4, you can also see that some items are not defined in any particular group. It is possible to mix properties defined from within a group with those that are not. The items not defined in a group in Listing 9-4 can still be accessed in the manner illustrated previously:

```
Label1.Text = Profile.FirstName
```

Now, using personalization groups, you can access your defined items in a logical manner using nested namespaces:

```
Label1.Text = Profile.MemberDetails.DateJoined  
  
Label2.Text = Profile.FamilyDetails.MarriedStatus
```

From this example, you can see that two separate items from each of the defined personalization groups that were defined were accessed in a logical manner. From the defined properties, you can see that each of the groups has a property with the same name — `Location`. This is possible because you are using personalization groups. With this structure, it is now possible to get at each of the `Location` properties by specifying the appropriate group:

```
Label1.Text = Profile.MemberDetails.Location  
  
Label2.Text = Profile.FamilyDetails.Location
```

Defining types for personalization properties

By default, when you store personalization properties, you store them as type `String`. It is quite easy, however, to change the type to something else through configurations within the `web.config` file. To define the name of the personalization property along with its type, you use the `Type` attribute, as shown in Listing 9-5.

Listing 9-5: Defining types for personalization properties

```
<properties>  
  
  <add name="FirstName" type="System.String" />  
  <add name="LastName" type="System.String" />  
  <add name="LastVisited" type="System.DateTime" />  
  <add name="Age" type="System.Integer" />  
  <add name="Member" type="System.Boolean" />  
  
</properties>
```


The first two properties, `FirstName` and `LastName`, are cast as type `String`. This isn't actually required. Even if you omitted this step, they would still be cast as type `String` because that is the default type. The next personalization property is the `LastVisited` property, which is defined as type `System.DateTime` and used to store the date and time of the end user's last visit to the page. Beyond that, you can see the rest of the personalization properties are defined using a specific .NET data type.

This is the preferred approach because it gives you type-checking capabilities as you code your application and use the personalization properties you have defined.

Using custom types

As you can see from the examples that show you how to define types for the personalization properties, it is quite simple to define types that are available in the .NET Framework. Items such as `System.Integer`, `System.String`, `System.DateTime`, `System.Byte`, and `System.Boolean` are easily defined within the `web.config` file. But how do you go about defining complex types?

Personalization properties that utilize custom types are just as easy to define as personalization properties that use simple types. Custom types give you the capability to store complex items such as shopping cart information or other status information from one use of the application to the next. Listing 9-6 first shows a class, `ShoppingCart`, which you use later in one of the personalization property definitions.

Listing 9-6: Creating a class to use as a personalization type

VB

```
<Serializable()> _
Public Class ShoppingCart
    Private PID As String
    Private CompanyProductName As String
    Private Number As Integer
    Private Price As Decimal
    Private DateAdded As DateTime

    Public Property ProductID() As String
        Get
            Return PID
        End Get
        Set(ByVal value As String)
            PID = value
        End Set
    End Property

    Public Property ProductName() As String
        Get
            Return CompanyProductName
        End Get
        Set(ByVal value As String)
            CompanyProductName = value
        End Set
    End Property

    Public Property NumberSelected() As Integer
        Get
```

```

        Return Number
    End Get
    Set(ByVal value As Integer)
        Number = value
    End Set
End Property

Public Property ItemPrice() As Decimal
    Get
        Return Price
    End Get
    Set(ByVal value As Decimal)
        Price = value
    End Set
End Property

Public Property DateItemAdded() As DateTime
    Get
        Return DateAdded
    End Get
    Set(ByVal value As DateTime)
        DateAdded = value
    End Set
End Property
End Class

```

C#

```

using System;

[Serializable]
public class ShoppingCart
{
    private string PID;
    private string CompanyProductName;
    private int Number;
    private decimal Price;
    private DateTime DateAdded;

    public ShoppingCart() {}

    public string ProductID
    {
        get { return PID; }
        set { PID = value; }
    }

    public string ProductName
    {
        get { return CompanyProductName; }
        set { CompanyProductName = value; }
    }

    public int NumberSelected
    {
        get { return Number; }
    }
}

```

(continued)

Listing 9-6: *(continued)*

```
        set { Number = value; }
    }

    public decimal ItemPrice
    {
        get { return Price; }
        set { Price = value; }
    }

    public DateTime DateItemAdded
    {
        get { return DateAdded; }
        set { DateAdded = value; }
    }
}
```

This simple shopping cart construction can now store the end user's shopping cart basket as she moves around on an e-commerce site. The basket can even be persisted when the end user returns to the site at another time.

Take a look at how you would specify from within the `web.config` file that a personalization property is this complex type. This is illustrated in Listing 9-7.

Listing 9-7: Using complex types for personalization properties

```
<properties>

  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" />
  <add name="Age" type="System.Integer" />
  <add name="Member" type="System.Boolean" />
  <add name="Cart" type="ShoppingCart" serializeAs="Binary" />

</properties>
```

Just as the basic data types are stored in the personalization data stores, this construction allows you to easily store custom types and to have them serialized into the end data store in the format you choose. In this case, the `ShoppingCart` object is serialized into a binary object in the data store. The `SerializeAs` attribute can take the values defined in the following list:

- ❑ **Binary:** Serializes and stores the object as binary data within the chosen data store.
- ❑ **ProviderSpecific:** Stores the object based upon the direction of the provider. This simply means that instead of the personalization engine determining the serialization of the object, it is simply left up to the data store.
- ❑ **String:** The default setting. Stores the personalization properties as a string inside the chosen data store.
- ❑ **XML:** Takes the object and serializes it into an XML format before storing it in the chosen data store.

Providing default values

In addition to defining the types of personalization properties, you can also define their default values. The personalization properties you create do not have a value, but you can easily change this using the `DefaultValue` attribute of the `<add>` element. Defining default values is illustrated in Listing 9-8.

Listing 9-8: Defining default values for personalization properties

```
<properties>

  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" />
  <add name="Age" type="System.Integer" />
  <add name="Member" type="System.Boolean" defaultValue="False" />

</properties>
```

From this example, you can see that only one of the personalization properties is provided with a default value. The last personalization property, `Member`, in this example, is given a default value of `False`. This means that when you add a new end user to the personalization property database, `Member` is defined instead of remaining a blank value.

Making personalization properties read-only

It is also possible to make personalization properties read-only. To do this, you simply add the `readOnly` attribute to the `<add>` element:

```
<add name="StartDate" type="System.DateTime" readOnly="True" />
```

To make the personalization property a read-only property, you give the `readOnly` attribute a value of `True`. By default, this property is set to `False`.

Anonymous Personalization

A great new feature in ASP.NET 2.0 enables anonymous end users to utilize the personalization features it provides. This is important if a site requires registration of some kind. In these cases, end users do not always register for access to the greater application until they have first taken advantage of some of the basic services. For instance, many e-commerce sites allow anonymous end users to shop a site and use the site's shopping cart before the shoppers register with the site.

Enabling anonymous identification of the end user

By default, anonymous personalization is turned off because it consumes database resources on popular sites. Therefore, one of the first steps to allow anonymous personalization is to turn on this feature using a setting in the `web.config` file.

As shown in Listing 9-9, you can turn on anonymous identification to enable the personalization engine to identify the unknown end users.

Listing 9-9: Allowing anonymous identification

```
<configuration>
  <system.web>

    <anonymousIdentification enabled="True" />

  </system.web>
</configuration>
```

To enable anonymous identification of the end users who might visit your applications, you add an `<anonymousIdentification>` element to the `web.config` file within the `<system.web>` nodes. Then within the `<anonymousIdentification>` element, you use the `Enabled` attribute and set its value to `True`. Remember that by default, this value is set to `False`.

When anonymous identification is turned on, ASP.NET uses a unique identifier for each anonymous user who comes to the application. This identifier is sent with each and every request, although after the end user becomes authenticated by ASP.NET, the identifier is removed from the process.

For an anonymous user, information is stored by default as a cookie on the end user's machine. Additional information (the personalization properties that you enable for anonymous users) is stored in the specified data store on the server.

Changing the name of the cookie for anonymous identification

Cookies are used by default under the cookie name `.ASPXANONYMOUS`. You can change the name of this cookie from the `<anonymousIdentification>` element in the `web.config` file by using the `cookieName` attribute, as shown in Listing 9-10.

Listing 9-10: Changing the name of the cookie

```
<configuration>
  <system.web>

    <anonymousIdentification
      enabled="True"
      cookieName=".ASPxEvjWebApplication" />

  </system.web>
</configuration>
```

Changing the length of time the cookie is stored

Also, by default, the cookie stored on the end user's machine is stored for 100,000 minutes (which is almost 70 days). If you want to change this value, you do it within this `<anonymousIdentification>` element through the use of the `cookieTimeout` attribute, as shown in Listing 9-11.

Listing 9-11: Changing the length of time the cookie is stored

```
<configuration>
  <system.web>

    <anonymousIdentification
      enabled="True"
      cookieTimeout="1440" />

  </system.web>
</configuration>
```

In this case, I changed the `cookieTimeout` value to 1440 — meaning 1,440 minutes (or one day). This would be ideal for a shopping cart for which you don't want to persist the identification of the end user too long.

Changing how the identifiers are stored

Although anonymous identifiers are stored through the use of cookies, you can also easily change this. Cookies are, by far, the preferred way to achieve identification, but you can also do it without the use of cookies. Other options include using the URI or device profiles. Listing 9-12 shows an example of using the URI to place the identifiers.

Listing 9-12: Specifying how cookies are stored

```
<configuration>
  <system.web>

    <anonymousIdentification
      enabled="True"
      cookieless="UseUri" />

  </system.web>
</configuration>
```

Besides `UseUri`, other options include `UseCookies`, `AutoDetect`, and `UseDeviceProfile`. The following list reviews each of the options:

- ❑ `UseCookies`: This is the default setting. If you set no value, ASP.NET assumes this is the value. `UseCookies` means that a cookie is placed on the end user's machine for identification.
- ❑ `UseUri`: This value means that a cookie *will not* be stored on the end user's machine, but instead the unique identifier will be munged within the URL of the page. This is the same approach used for cookieless sessions in ASP.NET 1.0/1.1. Although this is great if developers want to avoid sticking a cookie on an end user's machine, it does create strange looking URLs and can be an issue when an end user bookmarks pages for later retrieval.
- ❑ `AutoDetect`: Using this value means that you are letting the ASP.NET engine decide whether to use cookies or use the URL-approach for the anonymous identification. This is done on a per-user basis and performs a little worse than the other two options. ASP.NET must check the end user before deciding which approach to use. My suggestion is to use `AutoDetect` instead of

UseUri if you absolutely must allow for end users who have cookies turned off (which is rare these days).

- ❑ UseDeviceProfile: Configures the identifier for the device or browser that is making the request.

Looking at how the anonymous identifiers are stored

In order to make the anonymous identifiers unique, a globally unique GUID is used. You can also now grab hold of this unique identifier for your own use. In order to retrieve the GUID, the Request object has been enhanced with an AnonymousId property. The AnonymousId property returns a value of type String, which can be used in your code as shown here:

```
Label1.Text = Request.AnonymousId
```

Working with anonymous identification events

In working with the creation of anonymous users, be aware of two important events that can be used for managing the process:

- ❑ AnonymousIdentification_OnCreate
- ❑ AnonymousIdentification_OnRemove

By using the AnonymousIdentification_OnCreate event, you can work with the identification of the end user as it occurs. For instance, if you do not want to use GUIDs for uniquely identifying the end user, you can change the identifying value from this event instead.

To do so, create the event using the event delegate of type AnonymousIdentificationEventArgs, as illustrated in Listing 9-13.

Listing 9-13: Changing the unique identifier of the anonymous user

VB

```
Public Sub AnonymousIdentification_OnCreate(ByVal sender As Object, _  
    ByVal e As AnonymousIdentificationEventArgs)  
  
    e.AnonymousId = "Bubbles " & DateTime.Now()  
  
End Sub
```

C#

```
public void AnonymousIdentification_OnCreate(object sender,  
    AnonymousIdentificationEventArgs e)  
{  
    e.AnonymousId = "Bubbles " + DateTime.Now();  
}
```

The AnonymousIdentificationEventArgs event delegate exposes an AnonymousId property that assigns the value used to uniquely identify the anonymous user. Now, instead of a GUID to uniquely identify the anonymous user as

```
d13fafec-244a-4d21-9137-b213236ebedb
```

the `AnonymousID` property is changed within the `AnonymousIdentification_OnCreate` event to

Bubbles 6/10/2004 2:07:33 PM

The `AnonymousIdentification_OnRemove` event also employs an event delegate of type `AnonymousIdentificationEventArgs` that is used immediately prior to migrating anonymous users to registered users. Note that the `AnonymousId` property of the `Request` object is still accessible at this point.

Anonymous options for personalization properties

Now that the capability to work with anonymous users is in place, you have to specify which personalization properties you want to enable for anonymous users. This is also done through the `web.config` file by adding the `allowAnonymous` attribute to the `<add>` element (see Listing 9-14).

Listing 9-14: Turning on anonymous capabilities personalization properties

```
<properties>

  <add name="FirstName" type="System.String" />
  <add name="LastName" type="System.String" />
  <add name="LastVisited" type="System.DateTime" allowAnonymous="true" />
  <add name="Age" type="System.Integer" />
  <add name="Member" type="System.Boolean" />

</properties>
```

In this example, the `LastVisited` property is set to allow anonymous users by setting the `allowAnonymous` attribute to `True`. Because this is the only property that works with anonymous users, the rest of the defined properties do not store information for these types of users.

Migrating Anonymous Users

When working with anonymous users, you must be able to migrate anonymous users to registered users. For instance, after an end user fills a shopping cart, she can then register on the site to purchase the items. At this moment, she switches from an anonymous user to a registered user.

For this reason, ASP.NET 2.0 provides a `Profile_MigrateAnonymous` event enabling you to migrate anonymous users to registered users. The `Profile_MigrateAnonymous` event requires an event delegate of type `ProfileMigrateEventArgs`. It is placed either in the page that deals with the migration or within the `Global.asax` file (if it can be used from anywhere within the application). The use of this event is illustrated in Listing 9-15.

Listing 9-15: Migrating anonymous users for particular personalization properties

```
VB
Public Sub Profile_MigrateAnonymous(ByVal sender As Object, _
    ByVal e As ProfileMigrateEventArgs)
```

(continued)

Listing 9-15: *(continued)*

```
        Profile.LastVisited = Profile.GetProperty(e.AnonymousId).LastVisited
    End Sub
C#
    public void Profile_MigrateAnonymous(object sender,
        ProfileMigrateEventArgs e)
    {
        Profile.LastVisited = Profile.GetProperty(e.AnonymousId).LastVisited;
    }
```

From this example, you populate the new `Profile` property with the old property. You get at the old property of the anonymous user by using the `GetProperty` property, which takes a parameter of the ID of the anonymous user. From the `Profile_MigrateAnonymous` event, you still have access to the `AnonymousId` property, which you can retrieve from the event delegate — `ProfileMigrateEventArgs`.

Listing 9-15 shows how to migrate a single personalization property from an anonymous user to the new registered user. In addition to migrating single properties, you also must migrate properties that come from personalization groups. This is shown in Listing 9-16.

Listing 9-16: Migrating anonymous users for items in personalization groups**VB**

```
Public Sub Profile_MigrateAnonymous(ByVal sender As Object, _
    ByVal e As ProfileMigrateEventArgs)

    Dim au As HttpProfile = Profile.GetProfile(e.AnonymousId)

    If au.MemberDetails.DateJoined <> "" Then
        Profile.MemberDetails.DateJoined = DateTime.Now().ToString()
        Profile.FamilyDetails.MarriedStatus = au.FamilyDetails.MarriedStatus
    End If

End Sub
```

C#

```
public void Profile_MigrateAnonymous(object sender,
    ProfileMigrateEventArgs e)
{
    HttpProfile au = Profile.GetProfile(e.AnonymousId);

    if (au.MemberDetails.DateJoined != String.Empty) {
        Profile.MemberDetails.DateJoined = DateTime.Now.ToString();
        Profile.FamilyDetails.MarriedStatus = au.FamilyDetails.MarriedStatus;
    }
}
```

Using this event either in the page or in the `Global.asax` file enables you to logically migrate anonymous users as they register themselves with your applications. The migration process also allows you to pick and choose which items you migrate and to change the values as you wish.

Personalization Providers

As shown in Figure 9-1 earlier in the chapter, the middle tier of the Personalization model, the Personalization Services layer, communicates with a series of default data providers. By default, the Personalization model uses Microsoft Access for storing the personalization properties that you define. You are not limited to just this type of data store, however. Other built-in data providers include a Microsoft SQL Server data provider that enables you to work with Microsoft SQL Server 7.0, 2000, and SQL Yukon. Besides the Microsoft Access and SQL Server data providers, the architecture allows you to create your own data providers if one of these data stores doesn't fit your requirements.

Working with the Access personalization provider

The first data provider is the Microsoft Access data provider. This is the default provider used by the personalization system provided by ASP.NET. When used with Visual Studio 2005, the IDE places the `AspNetDB.mdb` file within your application's Data folder.

As you look through the `machine.config` file, note the sections that deal with how the personalization engine works with the Microsoft Access database. In the first reference to the Access file that it works with, you find a connection string to this file (shown in Listing 9-17) within the `<connections>` section located in the `<system.web>` section of the file.

Listing 9-17: The connection string to the Access file in the machine.config file

```
<configuration>
  <system.web>

    <connectionStrings>
      <add name="AccessFileName" connectionString="~/DATA\ASPNetDB.mdb" />
    </connectionStrings>

  </system.web>
</configuration>
```

In this example, you see that a connection string with the name of `AccessFileName` has been defined. The location of the file, specified by the `connectionString` attribute, points to the relative path of the file. This means that in every application you build that utilizes the new personalization capabilities, the default Access provider should be located in the application's Data folder and have the name of `ASPNetDB.mdb`.

The Access file's connection string is specified through the `AccessFileName` declaration within the `<connectionStrings>` section. You can see the Personalization engine's reference to this in the `<profile>` section within the `machine.config` file. The `<profile>` section here includes a subsection listing all the providers available to the Personalization engine. This is shown in Listing 9-18.

Listing 9-18: The Access data provider

```
<configuration>
  <system.web>

    <profile enabled="true" defaultProvider="AspNetAccessProvider"
```

(continued)

Listing 9-18: *(continued)*

```
inherits="System.Web.Profile.HttpProfileBase, System.Web,
Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
  <providers>
    <add name="AspNetAccessProvider"
      type="System.Web.Profile.AccessProfileProvider, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="AccessFileName" applicationName="/"
      description="Stores and retrieves profile data from the local
      Microsoft Access database file" />
  </providers>
</profile>

</system.web>
</configuration>
```

From this, you can see that a provider is added by using the `<add>` element. Within this element, the `connectionStringName` attribute points to what was declared in the `<connectionString>` attribute from Listing 9-17.

To specify an entirely different Microsoft Access file other than the one specified in the `machine.config` file, first create a connection string which points to an Access file that is a templated version of the `AspNetDB.mdb` file. You can find this template file at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\ASPNetdb_Template.mdb`. Just copy this file and locate the copy where you want. At this point, you can use the `<connectionString>` to point to this new Access file. If you change these values in the `machine.config` file, all the ASP.NET applications that reside on the server use this Access file. If you make the changes only to the `web.config` file, however, only the application that is using this particular `web.config` file uses this new data store. Other applications on the server remain unchanged.

Working with the SQL Server personalization provider

Although the Access personalization provider is the default provider when you work with the personalization framework, when you work with larger applications that require the factors of performance and reliability, you want to use the SQL Server personalization provider. If this data store is available, you should always try to use this option instead of the default Access personalization provider.

If you worked with the Access personalization provider I explained earlier, you probably found it easy to work with. It works with the personalization provider right out of the box — without any set up or configuration on your part. Using the SQL Server personalization provider is a bit of a different story. Although it is not difficult to work with, you must set up and configure your SQL Server before using it.

ASP.NET 2.0 provides three ways to set up and configure SQL Server for the personalization framework. The first way is through the Web Administration Tool, which is explained in detail in Chapter 14. The second way is through the ASP.NET SQL Server Setup Wizard, and the last method is by running some of the SQL Server scripts provided with the .NET Framework 2.0.

Using the ASP.NET SQL Server Setup Wizard

The ASP.NET SQL Server Setup Wizard is an easy-to-use tool that facilitates setup of the SQL Server to work with the personalization framework. The Setup Wizard provides two paths to set up the database — through the use of a command-line tool or through the use of a provided GUI tool.

The ASP.NET SQL Server Setup Wizard command-line tool

The command-line version of the Setup Wizard gives the developer optimal control over how the database is created. Working from the command-line using this tool is not difficult, so don't be intimidated by it.

You can get at the actual tool, `aspnet_regsql.exe`, from the Visual Studio Command Prompt. You can find this command prompt at Start ⇨ All Programs ⇨ Microsoft Visual Studio Whidbey ⇨ Visual Studio Tools ⇨ Visual Studio Command Prompt. At the command prompt, type `aspnet_regsql.exe -?` to get a list of all the available command-line options at your disposal for working this tool.

The following table describes some of the available options for setting up your SQL Server instance to work with the personalization framework.

Command Option	Description
-?	Displays a list of available option commands.
-W	Uses the Wizard mode. This uses the default installation if no other parameters are used.
-S <server>	Specifies the SQL Server instance to work with.
-U <login>	The username to log in to SQL Server. If used, you also use the -P command.
-P <password>	The password to use for logging in to SQL Server. If used, you also use the -U command.
-E	Provides instructions to use the current Windows credentials for authentication.
-C	Specifies the connection string for connecting to SQL Server. If used, you can avoid using the -U and -P commands because they are specified in the connection string itself.
-A all	Adds support for all the available SQL Server operations provided by ASP.NET 2.0 including membership, role management, profiles, site counters, and page/control personalization.
-A p	Adds support for working with profiles.
_R all	Removes support for all the available SQL Server operations that have been previously installed. These include membership, role management, profiles, site counters, and page/control personalization.
-R p	Removes support for the profile capability from SQL Server.
-d <database>	Specifies the database name to use with the application services. If you don't specify a name of a database, <code>aspnetdb</code> is used.
/sqlexportonly <filename>	Instead of modifying an instance of a SQL Server database, use this command in conjunction with the other commands to generate a SQL script that adds or removes the features specified. This command creates the scripts in a file by the name specified in the command.

Chapter 9

To modify SQL Server to work with the personalization provider using this command-line tool, you enter a command such as the following:

```
aspnet_regsql.exe -A p -E
```

After you enter the preceding command, the command-line tool creates the profile features required. The results are shown in the tool itself as you see in Figure 9-4.



Figure 9-4

When this action is completed, you can see that, in fact, a new table, aspnetdb, has been created in the SQL Server Enterprise Manager. It now contains the appropriate tables for working with the personalization framework (see Figure 9-5).

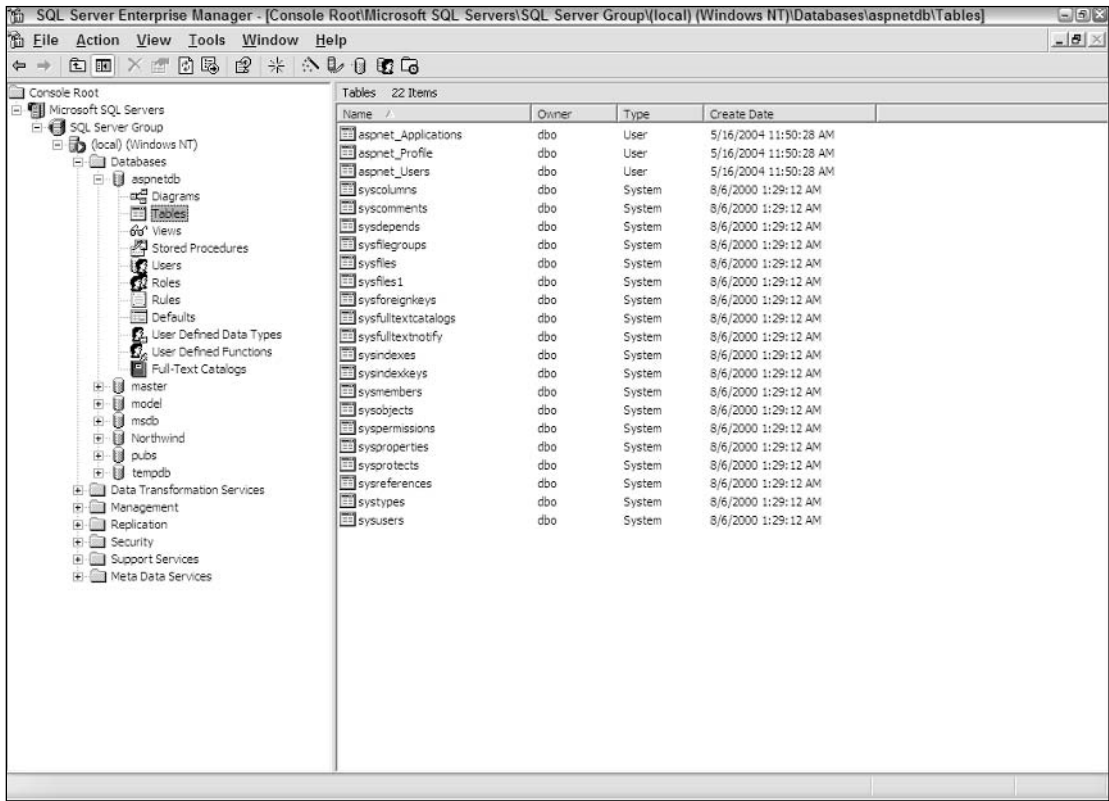


Figure 9-5

The ASP.NET SQL Server Setup Wizard GUI tool

Instead of working with this tool through the command-line, another option is to work with the same wizard through a GUI version of it. To get at the GUI version, type the following at the Visual Studio Command Prompt:

```
aspnet_regsql.exe
```

At this point, the ASP.NET SQL Server Setup Wizard welcome screen appears, as shown in Figure 9-6.

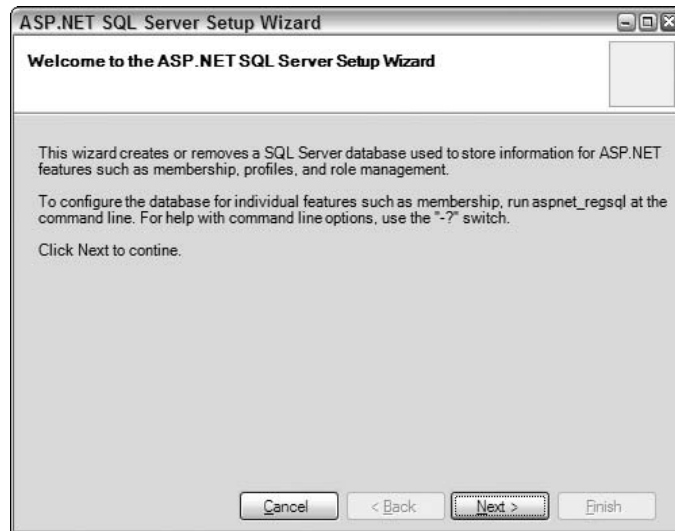


Figure 9-6

Clicking the Next button gives you a new screen that provides you with two options — one to install management features into SQL Server and the other to remove them (see Figure 9-7).



Figure 9-7

Chapter 9

From here, choose the Configure SQL Server for ASP.NET SQL Server Features and click the Next button. The third screen (see Figure 9-8) asks for the login credentials to SQL Server and the name of the database to perform the operations. When you pull it up, the Database option is `<default>` — meaning that the wizard creates a database called `aspnetdb`. If you want to choose a different folder, such as the application's database, choose the appropriate option.



Figure 9-8

After you have made your server and database selections, click Next. The screen shown in Figure 9-9 asks you to confirm your settings. If everything looks correct, click the Next button — otherwise click Back and correct your settings.

When you're finished, you get a notification that everything was set up correctly.

Using SQL scripts to install personalization features

Another option is to use the same SQL scripts that these tools and wizards use. If you look at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\`, from this location, you see the install and remove scripts — `InstallPersonalization.sql` and `UninstallPersonalization.sql`. Running these scripts provides your database with the tables needed to run the personalization framework.

Setting up your application to use a SQL Server personalization provider

Now SQL Server is set up to work with the personalization capabilities provided by ASP.NET 2.0. The personalization framework understands how to work with SQL through settings that are in the `machine.config` or `web.config` files.



Figure 9-9

If you look in the `machine.config` file, you find the connection string to SQL Server is specified in the `<connectionStrings>` section of the document, as shown in Listing 9-19.

Listing 9-19: The connection string to SQL Server in the `machine.config` file

```
<configuration>
  <system.web>

    <connectionStrings>
      <add name="LocalSqlServer"
        connectionString="data source=127.0.0.1;Integrated Security=SSPI" />
    </connectionStrings>

  </system.web>
</configuration>
```

You may want to change the values provided if you are working with a remote instance of SQL Server rather than an instance that resides on the same server as the application. Changing this value in the `machine.config` file changes how each and every ASP.NET application uses this provider.

After the connection string is set up accordingly, look further in the `<providers>` section of the `<profile>` element. You see the settings for SQL Server, as shown in Listing 9-20.

Listing 9-20: The SQL Server data provider

```
<configuration>
  <system.web>

    <profile enabled="true" defaultProvider="AspNetAccessProvider"
```

(continued)

Listing 9-20: *(continued)*

```
inherits="System.Web.Profile.HttpProfileBase, System.Web,
Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
  <providers>
    <add name="AspNetSqlProvider"
      type="System.Web.Profile.SqlProfileProvider,
      System.Web, Version=2.0.3600.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a" connectionStringName="LocalSqlServer"
      applicationName="/" description="Stores and retrieves profile data
      from the local Microsoft SQL Server database" />
  </providers>
</profile>

</system.web>
</configuration>
```

With this addition, SQL Server is now added as one of available providers to use with your applications. The name of this provider instance is `AspNetSqlProvider`. You can see that this instance also uses the connection string of `LocalSqlServer`, which was defined in Listing 9-19.

Now that the SQL Server provider is in place and ready to use, you have to make only a minor change to the `web.config` file of your application in order for your application to take advantage of what you have established. Listing 9-21 shows the `<profile>` section of the `web.config` file.

Listing 9-21: Using SQL Server as the provider in the web.config file

```
<configuration>
  <system.web>

    <profile inherits="System.Web.Profile.HttpProfileBase, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      defaultProvider="AspNetSqlProvider">

      <properties>

        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" />

      </properties>

    </profile>

  </system.web>
</configuration>
```

The only change necessary is to use the `defaultProvider` attribute and give it a value that is the name of the provider you want to use — in this case the SQL Server provider, `AspNetSqlProvider`. You could have also made this change to the `machine.config` file by changing the `<profile>` element as shown in Listing 9-22.

Listing 9-22: Using SQL Server as the provider in the machine.config file

```

<configuration>
  <system.web>

    ...

    <profile enabled="true" defaultProvider="AspNetSqlProvider"
      inherits="System.Web.Profile.HttpProfileBase, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">

      ...

    </profile>

    ...

  </system.web>
</configuration>

```

This change forces each and every application that resides on this server to use the SQL Server provider instead of the Access provider, unless this command is overridden in the application's `web.config` file.

Using multiple providers

You are not limited to using a single provider, such as the Access or SQL provider. Instead, you can use any number of providers. You can specify the personalization provider for each property defined. This means that you can use the default provider for most of the properties, whereas a few of them use an entirely different provider (see Listing 9-23).

Listing 9-23: Using different providers

```

<configuration>
  <system.web>

    <profile inherits="System.Web.Profile.HttpProfileBase, System.Web,
      Version=2.0.3600.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      defaultProvider="AspNetSqlProvider">

      <properties>

        <add name="FirstName" />
        <add name="LastName" />
        <add name="LastVisited" />
        <add name="Age" />
        <add name="Member" provider="AspNetAccessProvider" />

      </properties>

    </profile>

  </system.web>
</configuration>

```

From this example, you can see that a default provider is specified — `AspNetSqlProvider`. Unless specified otherwise, this provider is used. The only property that changes this setting is the property `Member`. The `Member` property uses an entirely different personalization provider. In this case, it employs the Access provider (`AspNetAccessProvider`) through the use of the `provider` attribute of the `<add>` element. With this attribute, you can define a specific provider for each and every property that is defined.

Summary

The new personalization capabilities provided by ASP.NET 2.0 make it incredibly easy to make your Web applications unique for all end users, whether they are authenticated or anonymous. This new system enables you to store everything from basic data types provided by the .NET Framework to custom types that you create. This system is more versatile and extensible than the use of `Session` or `Application` objects. The data is stored via a couple of built-in personalization providers that ship with ASP.NET. These providers include ones that connect with either Microsoft Access or Microsoft SQL Server.

10

Portal Frameworks and Web Parts

Internet and intranet applications have changed considerably since their introduction in the 1990s. Today's applications don't simply display the same canned information to every viewer; they do much more. Because of the wealth of information exposed to end users, Internet and intranet applications must integrate large amounts of customization and personalization into their offerings.

Web sites that provide a plethora of offerings give end users the option to choose which parts of the site they want to view and which parts they want to hide. Ideally, end users can personalize the pages, deciding for themselves the order in which the content appears on the page. They should be able to move items around on the page as if it were a design surface.

After pages are customized and established, the end user needs the capability to export his settings for storage. You certainly wouldn't want to allow an end user to highly customize a page or a series of pages in your portal and then force him to reapply the settings each time he visits the site. Instead, developers need to take these setting points and move the items to a data store for later exposure.

Adding this kind of functionality is expensive — *expensive* in the sense that it can take a considerable amount of work on the part of the developer. Before ASP.NET 2.0, the developer had to build a personalization framework that would be used by each page requiring the functionality. This type of work is error prone and difficult to achieve, which is why in most cases it wasn't developed.

But wait. . . .

Introducing Web Parts

To make it easier to retain the page customization settings applied to your page by your application's end users, Microsoft has included Web Parts in this release of ASP.NET. Web Parts, part of the larger Portal Framework, are an outstanding way to build a modular Web site that can be customized with dynamically reapplied settings on a per-user basis. Web Parts are objects in the

Chapter 10

Portal Framework which the end user can open, close, minimize, maximize or move from one part of the page to another.

The Portal Framework enables you to build pages that contain multiple Web Parts — which are part of the ASP.NET server control framework and are used like any of the other ASP.NET server controls. This means that you can also extend Web Parts if necessary.

The components of the Portal Framework provide the means to build a truly dynamic Web site, whether that site is a traditional Internet site, an intranet site, a browser-based application, or any other typical portal.

When you first look at Web Parts in ASP.NET 2.0, it may remind you of Microsoft's SharePoint Portal Server. Be forewarned, however, that these two technologies are similar in name only. The ASP.NET team introduced Web Parts; the resulting Portal Framework is the basis of what is now also used by the SharePoint Portal Server, in addition to being offered in ASP.NET. Microsoft is simply creating singular technologies that can be used by other Microsoft offerings. In this process, Microsoft is trying to reach the Holy Grail of computing — *code reuse*!

The modular and customizable sites that you can build with the new Portal Framework enable you to put the Web page that is in view in several possible modes for the end user. The following list describes each of the available modes and what each means to the end user viewing the page:

- ❑ **Normal Mode:** Puts the page in a normal state, which means that the end user cannot edit or move sections of the page. This is the mode used for standard page viewing.
- ❑ **Edit Mode:** Enables end users to select particular sections on the page for editing. The selected section allows all types of editing capabilities from changing the part's title, the part's color, or even setting custom properties — such as specifying the end user's zip code to pull up a customized weather report.
- ❑ **Design Mode:** Enables end users to rearrange the order of the page's modular components. The end user can bring items higher or lower within a zone, delete items from a zone, or move items from one page zone to another.
- ❑ **Catalog Mode:** Displays a list of available sections (Web Parts) that can be placed in the page. Catalog Mode also allows the end user to select in which zone on the page the items should appear.

Figure 10-1 shows a screen shot of a portal utilizing the Portal Framework with the Edit Mode selected.

The Portal Framework is a comprehensive and thought-out framework that enables you to incorporate everything you would normally include in your ASP.NET applications. You can apply security using either Windows Authentication or Forms Authentication. This framework also enables you to leverage the other new aspects of ASP.NET 2.0, such as applying role management, personalization, and membership features to any portal that you build.

To understand how to build your own application on top of the new Portal Framework, start by creating a simple page that uses this new framework's utilities.

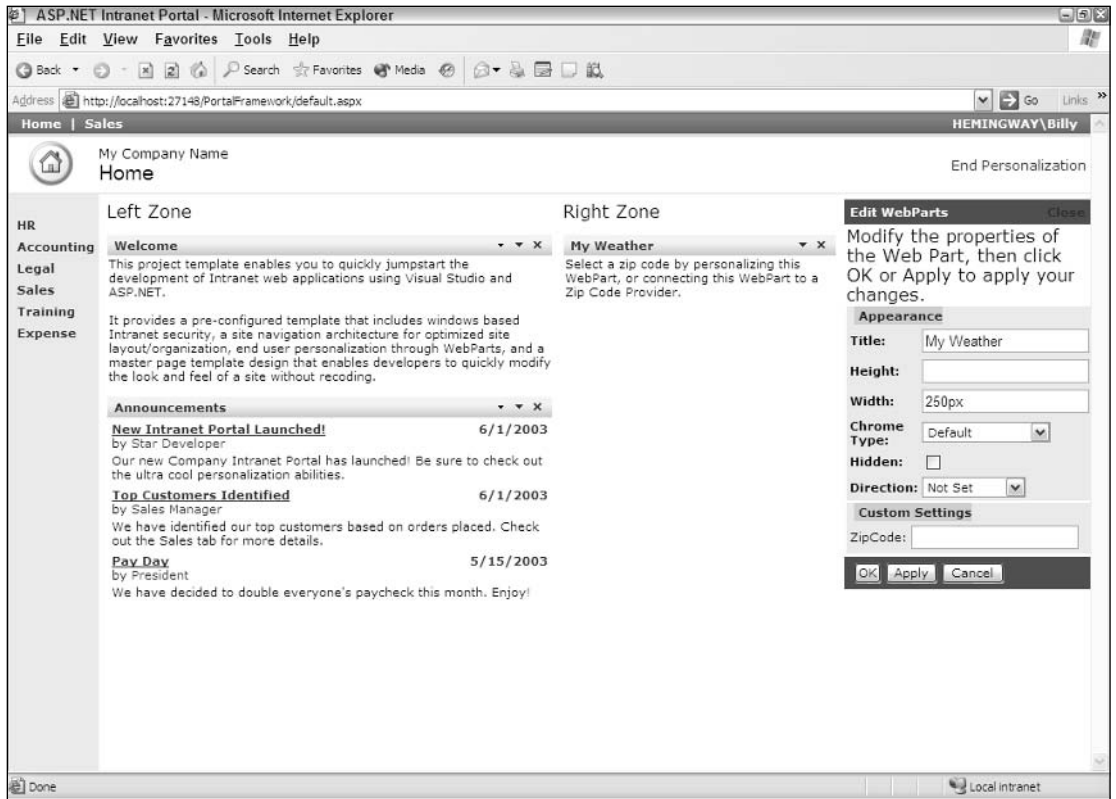


Figure 10-1

Building Dynamic and Modular Web Sites

As you begin using the new Portal Framework to build Web sites, note that the framework defines everything in *zones*. There are zones for laying out content and zones for editing content. The zones that a page might incorporate are managed by a Portal Framework manager — they don't have to be managed by you in any fashion — which makes working with this new Portal Framework a breeze.

This framework contains a lot of moving parts (pieces that are dependent upon each other), so I start at the beginning by examining the Portal Framework manager control: *WebPartManager*.

Introducing the *WebPartManager* control

The *WebPartManager* control is an ASP.NET server control that completely manages the state of the zones and the content placed in the zones on a per-user basis. This control, which has no visual aspect, can add and delete items contained within each zone of the page. The *WebPartManager* control can also

Chapter 10

manage the communications sometimes required between different elements contained in the zones. For example, you can pass a specific name/value pair from one item to another item within the same zone, or between items contained in entirely separate zones. The WebPartManager control provides the capabilities to make this communication happen.

The WebPartManager control is required to be in place on every page in your application that works with the Portal Framework. A single WebPartManager control does not manage an entire application; it manages on a per-page basis.

Listing 10-1 shows a WebPartManager control added to an ASP.NET page.

Listing 10-1: Adding a WebPartManager control to an ASP.NET page

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Web Parts Example</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:WebPartManager ID="Webpartmanager1" Runat="server">
      </asp:WebPartManager>
    </form>
  </body>
</html>
```

If you want to work from the design surface of Visual Studio 2005, you can drag and drop the WebPartManager control from the Toolbox to the design surface — but remember, it does not have a visual aspect and appears only as a gray box. You can find the WebPartManager control (and the other server controls that are part of the Portal Framework) in the Personalization section of the Toolbox, as shown in Figure 10-2.

Working with zone layouts

After you place the WebPartManager control, the next step is to create zones on the page on which you want to utilize the Portal Framework. You should give this step some thought because it contributes directly to the usability of the page you are creating. Web pages are constructed in a linear fashion — either horizontally or vertically. Web pages are managed in square boxes — usually through the use of tables that organize the columns and rows in which items appear on the page.

Web zones define specific rows or columns as individual content areas managed by the WebPartManager. For an example of a Web page that uses these zones, you can create a table similar to the one shown in Figure 10-3.



Figure 10-2

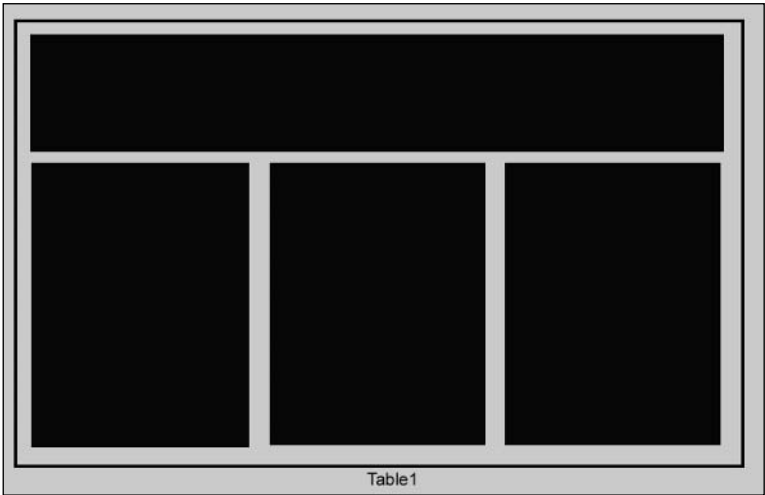


Figure 10-3

Chapter 10

The black sections in Figure 10-3 represent Web zones. The code used to produce the table is shown in Listing 10-2.

Listing 10-2: Creating multiple Web zones

```
<%@ Page Language="VB"%>
<%@ Register TagPrefix="myUserControl1" TagName="DailyLinksWebPart"
    Src="~/controls/DailyLinksWebPart.ascx" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Web Parts Example</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:WebPartManager ID="Webpartmanager1" Runat="server">
</asp:WebPartManager>
        <table cellpadding="5" border="1">
            <tr>
                <td colspan="3">
                    <h1>Bill Evjen's Web Page</h1>
                    <asp:WebPartZone ID="WebPartZone1" Runat="server"
                        LayoutOrientation="Horizontal">
                        <ZoneTemplate>
                            <asp:Label ID="Label1" Runat="server" Text="Label"
                                Title="Welcome to my web page!">
                                Welcome to the page!
                            </asp:Label>
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
            </tr>
            <tr valign="top">
                <td>
                    <asp:WebPartZone ID="WebPartZone2" Runat="server">
                        <ZoneTemplate>
                            <asp:DynamicImage ID="DynamicImage1" Runat="server"
                                ImageFile="~/kids.jpg" Width="150" Title="My Kids">
                            </asp:DynamicImage>
                            <myUserControl1:DailyLinksWebPart
                                ID="DailyLinksWebPart1" Runat="server"
                                Title="Daily Links" >
                            </myUserControl1:DailyLinksWebPart >
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
                <td>
                    <asp:WebPartZone ID="WebPartZone3" Runat="server">
                        <ZoneTemplate>
                            <asp:Calendar ID="Calendar1" Runat="server">
                            </asp:Calendar>
                        </ZoneTemplate>
                    </asp:WebPartZone>
                </td>
                <td><!-- Blank for now -->
```

```

        </td>
      </tr>
    </table>
  </form>
</body>
</html>

```

This page now has sections like the ones shown in Figure 10-3: a header section that runs horizontally and three vertical sections underneath the header. Running this page provides the result shown in Figure 10-4.

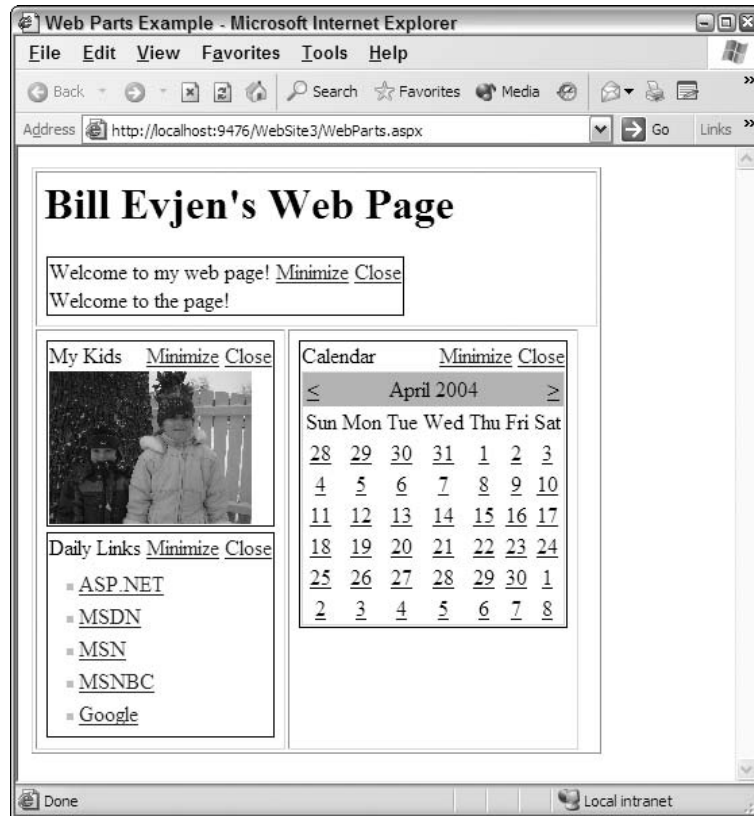


Figure 10-4

First, this page includes the `<asp:WebPartManager>` control that manages the items contained in the three zones on this page. Within the table, the `<asp:WebPartZone>` server control specifies three Web zones. You can declare each Web zone in one of two ways. You can use the `<asp:WebPartZone>` element directly in the code, or you can create the zones within the table by dragging and dropping `WebPartZone` controls onto the design surface at appropriate places within the table. In Figure 10-4, the table border width is intentionally turned on and set to 1 in order to show the location of the Web zones in greater detail. Figure 10-5 shows what the sample from Listing 10-2 looks like in the Design view of Visual Studio 2005.

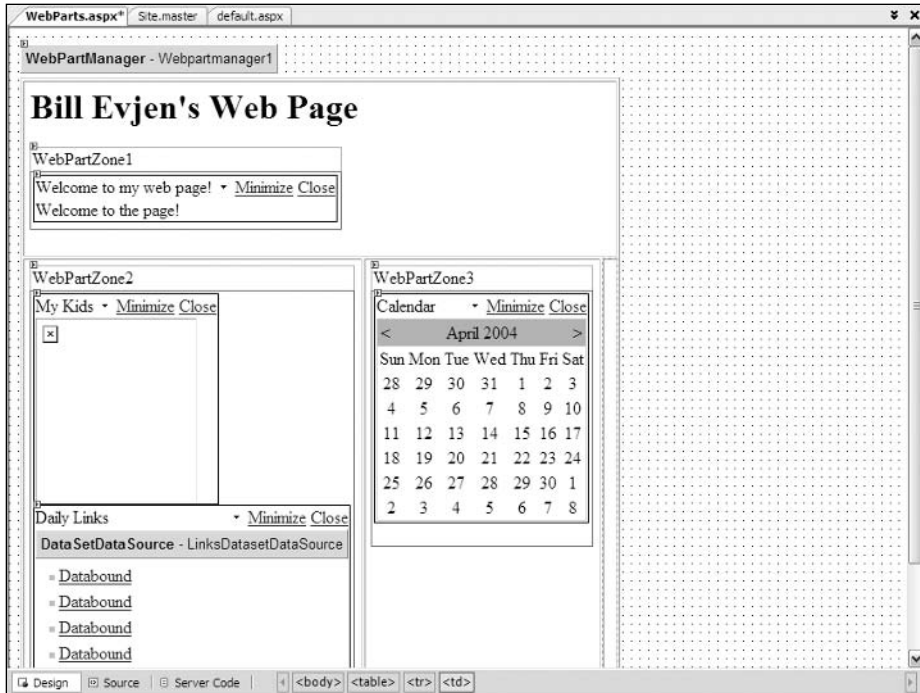


Figure 10-5

When using Visual Studio 2005, note that this IDE creates an Access file called `ASPNetDB.mdb` and stores it in the `Data` folder of your Web Project. This database file is where the Portal Framework stores all the customization points.

Now that you have seen the use of `WebPartZone` controls, which are managed by the `WebPartManager` control, the next section takes a closer look at the `WebPartZone` control itself.

Understanding the `WebPartZone` control

The `WebPartZone` control defines an area of items, or Web Parts, that can be moved, minimized, maximized, deleted, or added based on programmatic code or user preferences. When you drag and drop `WebPartZone` controls onto the design surface using Visual Studio 2005, the `WebPartZone` control is drawn at the top of the zone, along with a visual representation of any of the items contained within the zone.

You can place almost anything in one of the Web zones. For example, you can include

- ☐ HTML elements
- ☐ Raw text
- ☐ HTML server controls

- ☐ Web server controls
- ☐ User controls
- ☐ Custom controls

WebPartZone controls are declared like this:

```
<asp:WebPartZone ID="WebPartZone1" Runat="server"></asp:WebPartZone>
```

The LayoutOrientation attribute

The Web Parts declared within a WebPartZone control can be displayed either horizontally or vertically. By default, all the items are displayed vertically, but to display the items horizontally, you simply add the `LayoutOrientation` attribute to the `<asp:WebPartZone>` element:

```
<asp:WebPartZone ID="WebPartZone1" Runat="server"
  LayoutOrientation="Horizontal"></asp:WebPartZone>
```

The first row in the table from Listing 10-2 uses horizontal orientation, whereas the other two zones use the default vertical orientation.

The ZoneTemplate element

In order to include items within the templated WebPartZone control, you must include a `<ZoneTemplate>` element. The `ZoneTemplate` element encapsulates all the items contained within a particular zone. The order in which they are listed in the `ZoneTemplate` section is the order in which they appear in the browser until changed by the end user or by programmatic code. The sample `<ZoneTemplate>` section used earlier is illustrated here:

```
<asp:WebPartZone ID="WebPartZone2" Runat="server">
  <ZoneTemplate>
    <asp:DynamicImage ID="DynamicImage1" Runat="server"
      ImageFile="~/kids.jpg" Width="150" Title="My Kids">
    </asp:DynamicImage>
    <myUserControl1:DailyLinksWebPart
      ID="DailyLinksWebPart1" Runat="server"
      Title="Daily Links" >
    </myUserControl1:DailyLinksWebPart >
  </ZoneTemplate>
</asp:WebPartZone>
```

This zone contains two items — a dynamic image and a user control consisting of a collection of links that come from an XML file.

Default Web Part control elements

By default, when you generate a page using the code from Listing 10-2, you discover that you can exert only minimal control over the Web Parts themselves. In the Default view, which isn't the most artistic, you are only able to minimize or close a Web Part, which removes that particular Web Part from the screen.

Chapter 10

Figure 10-6 shows what the Web Part that contains the Calendar control looks like after you minimize it. Notice also that if you opt to close one of the Web Parts, the item completely disappears. You seem to have no way to make it come back — even if you shut down the page and restart it. This is by design — so don't worry. I show you how to get it back!

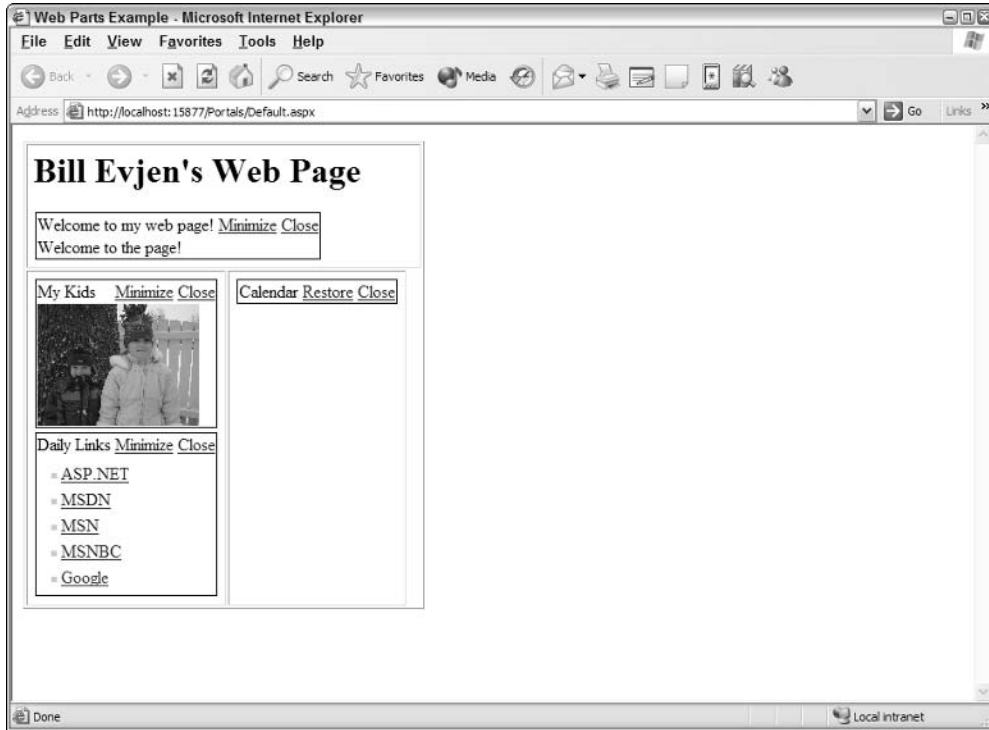


Figure 10-6

A few of the items included in the zones have new titles. By default, the title that appears at the top of the Web Part is the name of the control. For instance, you can see that the Calendar control is simply titled Calendar. If you add a Button control to the zone, it would simply be called Button. To give better and more meaningful names to the Web Parts that appear in a zone, you simply add a `Title` attribute to the control — just as you did with the `DynamicImage` control and the `User` control, which both appear on the page. In the example above, I rename the `DynamicImage` control `My Kids`, whereas I give the user control the `Title` value `Daily Links`.

Besides this little bit of default functionality, you can do considerably more with the Web Parts contained within this page, but you have to make some other additions, which I review next.

Explaining the WebPartPageMenu control

The WebPartPageMenu control is another management control that simplifies applying customization capabilities to your pages that utilize the Portal Framework. This control provides a menu on the page so that the end user can customize the page and then save his changes to the server.

The WebPartPageMenu control enables the end user to

- ❑ **Add new Web Parts to the page:** Includes Web Parts not displayed on the page by default and Web Parts that the end user has previously deleted. This aspect of the control works with the catalog capabilities of the Portal Framework, which is discussed shortly.
- ❑ **Enter the design mode for the page:** Enables the end user to drag and drop elements around the page. The end user can use this capability to change the order in which items appear in a zone or to move items from one zone to another.
- ❑ **Modify the Web Parts settings:** Enables the end user to customize aspects of the Web Parts, such as their appearance and behavior. It also allows the end user to modify any custom settings that developers apply to the Web Part.
- ❑ **Connect Web Parts on the page:** Enables the end user to make a connection between one or more Web Parts on the page. For instance, if the end user is working in a financial services application and enters a stock symbol into one of the Web Parts — by using a connection to another Web Part — a chart is changed or news appears based upon the variable defined in the first Web Part.

Building upon Listing 10-2, Listing 10-3 adds a WebPartPageMenu control to the table's header.

Listing 10-3: Adding a WebPartPageMenu control

```
<tr valign="top">
  <td colspan="2">
    <h1>Bill Evjen's Web Page</h1>
    <asp:WebPartZone ID="WebPartZone1" Runat="server"
      LayoutOrientation="Horizontal">
      <ZoneTemplate>
        <asp:Label ID="Label1" Runat="server" Text="Label"
          Title="Welcome to my web page!">
          Welcome to the page!
        </asp:Label>
      </ZoneTemplate>
    </asp:WebPartZone>
  </td>
  <td>
    <asp:WebPartPageMenu ID="Webpartpagemenu1" Runat="server">
    </asp:WebPartPageMenu>
  </td>
</tr>
```

This adds the WebPartPageMenu control to the top of the table as shown in Figure 10-7.

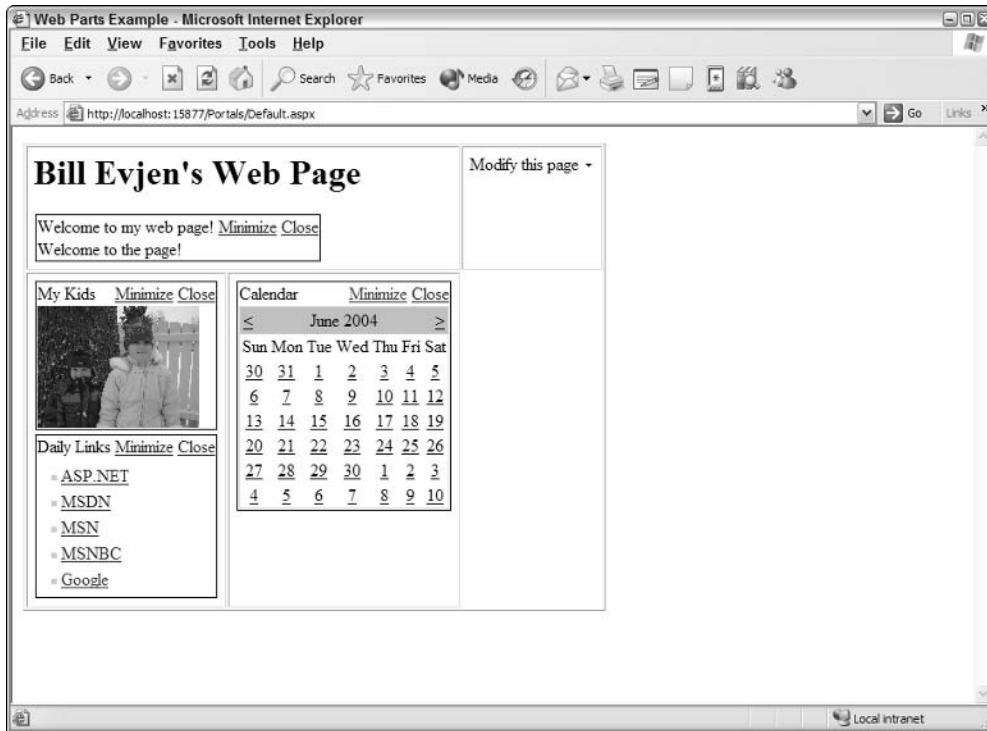


Figure 10-7

When the end user clicks on the link, a drop-down window of options appears, as shown in Figure 10-8.

The next section covers an important addition to the Portal Framework provided by the `WebPartPageMenu` control — the capability to Add Web Parts to This Page.

Adding Web Parts to a page

The end user has a built-in way to add Web Parts to the page by using the Portal Framework. The `WebPartPageMenu` control enables you to add Web Parts, but you also need to provide the end user with a list of items he can add. To do this, simply add a `Catalog Zone` to the last table cell in the bottom of the table. This is illustrated in the partial code example in Listing 10-4.

Listing 10-4: Adding a Catalog Zone

```
<tr valign="top">
  <td>
    <asp:WebPartZone ID="WebPartZone2" Runat="server">
      <ZoneTemplate>
        <asp:DynamicImage ID="DynamicImage1" Runat="server"
          ImageFile="~/kids.jpg" Width="150" Title="My Kids">
        </asp:DynamicImage>
        <myUserControl1:DailyLinksWebPart
          ID="DailyLinksWebPart1" Runat="server"
          Title="Daily Links" >
```

```

        </myUserControl1:DailyLinksWebPart >
    </ZoneTemplate>
</asp:WebPartZone>
</td>
<td>
    <asp:WebPartZone ID="WebPartZone3" Runat="server">
        <ZoneTemplate>
            <asp:Calendar ID="Calendar1" Runat="server">
            </asp:Calendar>
        </ZoneTemplate>
    </asp:WebPartZone>
</td>
<td>
    <asp:CatalogZone ID="Catalogzone1" Runat="server">
        <ZoneTemplate>
            <asp:PageCatalogPart ID="Pagecatalogpart1" Runat="server" />
        </ZoneTemplate>
    </asp:CatalogZone>
</td>
</tr>

```

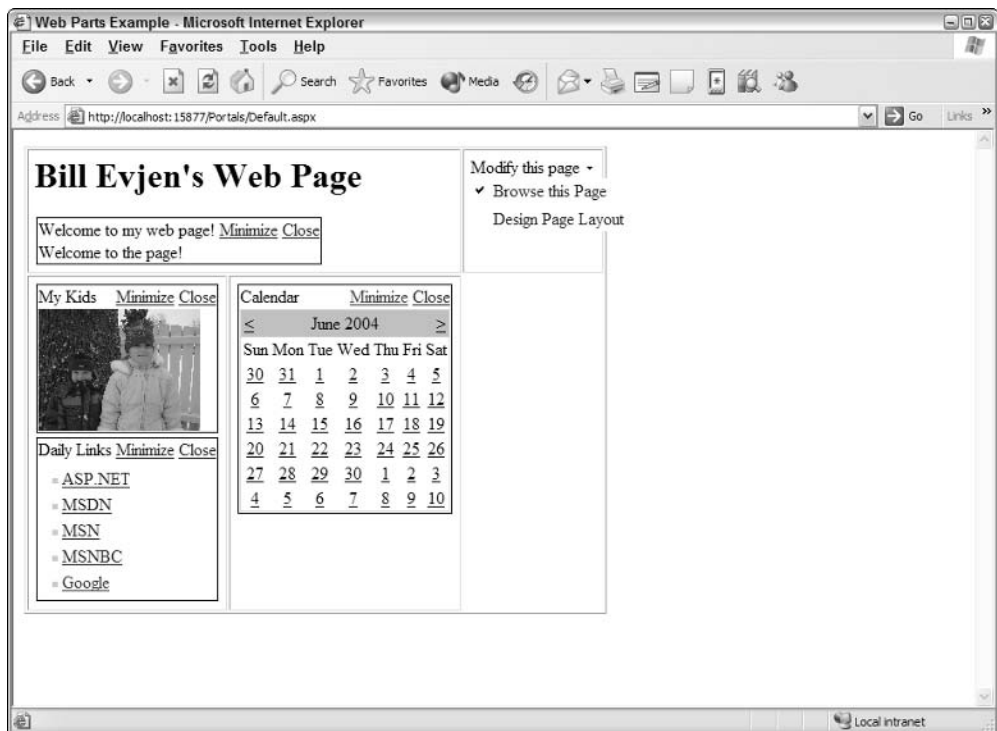


Figure 10-8

To add the capability for the end user to select items to place on the page after he selects the option in the WebPartPageMenu control, you must create a Catalog Zone with the `<asp:CatalogZone>` control. This is similar to creating a Web Part Zone, but the Catalog Zone is specifically designed to allow for categorization of items to be placed on the page. After the Catalog Zone is in place, the next step is to create a `<ZoneTemplate>` section within the Catalog Zone because this is a templated section. Inside the `<ZoneTemplate>` element is a single control — the PageCatalogPart control. If you run the page after adding the PageCatalogPart control, you see the results shown in Figure 10-9.

To get some items to appear in the list, delete one or more items from the page's default view and enter the catalog mode through the WebPartPageMenu control. At this point, you see the deleted Web Parts. The PageCatalogPart control contains a title and check box list of items that can be selected. The PageCatalogPart control also includes a drop-down list of all the available Web Part Zones on the page. You can place the selected Web Parts in this list. After you select the Web Parts and the appropriate zone, you click the Add button and the items appear in their specified locations.

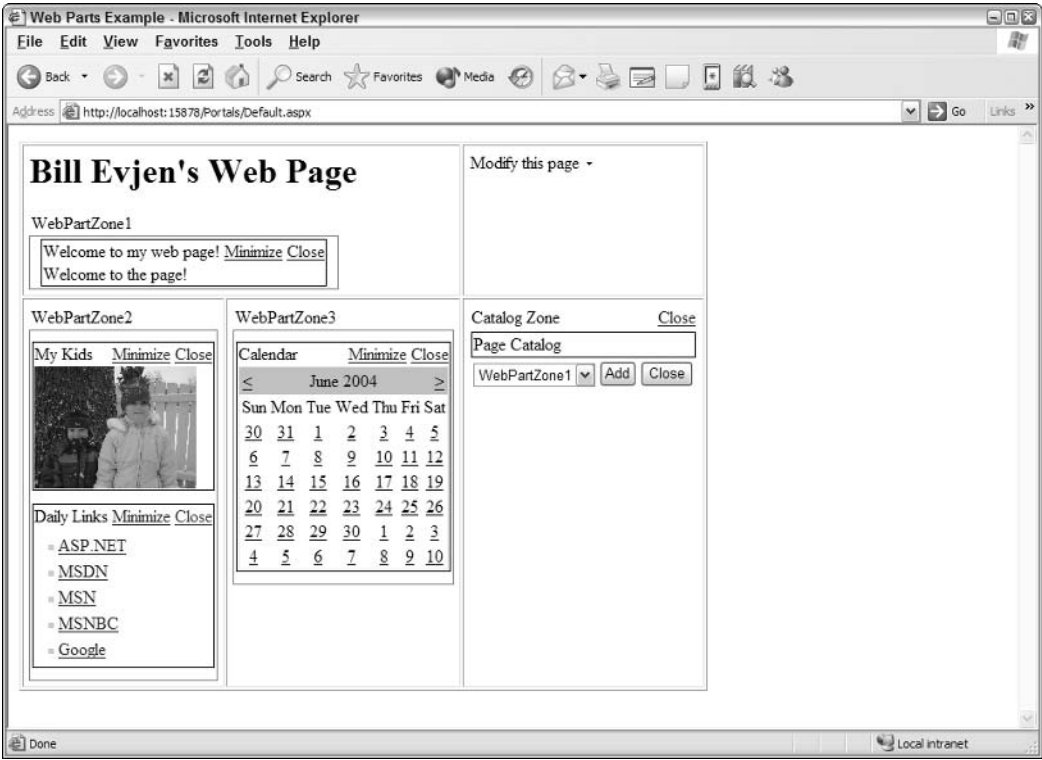


Figure 10-9

Moving Web Parts

Not only can the end user change the order in which Web Parts appear in a zone, he can also move Web Parts from one zone to another. By adding the WebPartPageMenu control, you have already provided the end user with this capability. He simply needs to enter a mode that allows for this type of movement.

The end user can move Web Parts by selecting Design Page Layout from the list. This is the second option in the list and changes the page so that the end user can see the zones defined on the page. This view is illustrated in Figure 10-10.

From this figure, you can see the three zones (`WebPartZone1`, `WebPartZone2`, and `WebPartZone3`). At this point, the end user can select one of the Web Parts and either change its order in the zone or move it to an entirely different zone on the page. To grab one of the Web Parts, the user simply clicks and holds the left mouse button on the title of the Web Part. When done correctly, a cross-hair appears, meaning that the user has grabbed hold of the Web Part and can drag it to another part of the page. While the end user drags the Web Part around the page, a visual representation of the item appears (see Figure 10-11). In this state, the Web Part is a bit transparent and its location in the state of the page is defined with a blue line. Releasing the left mouse button drops the Web Part at the blue line's location.

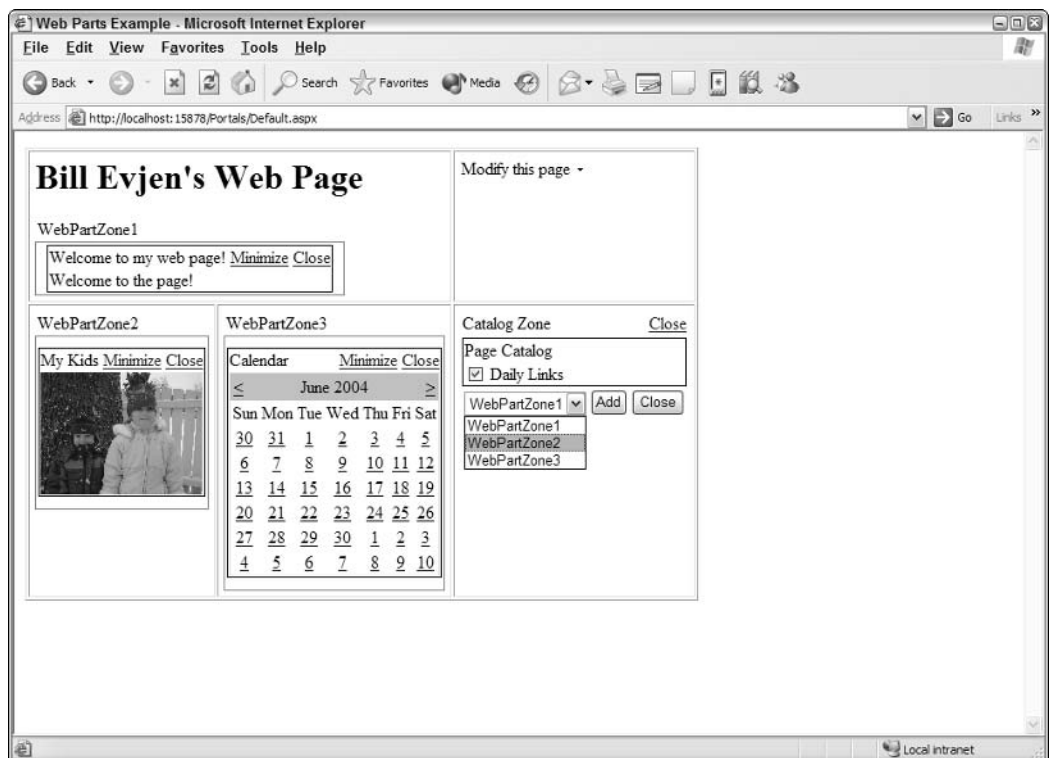


Figure 10-10

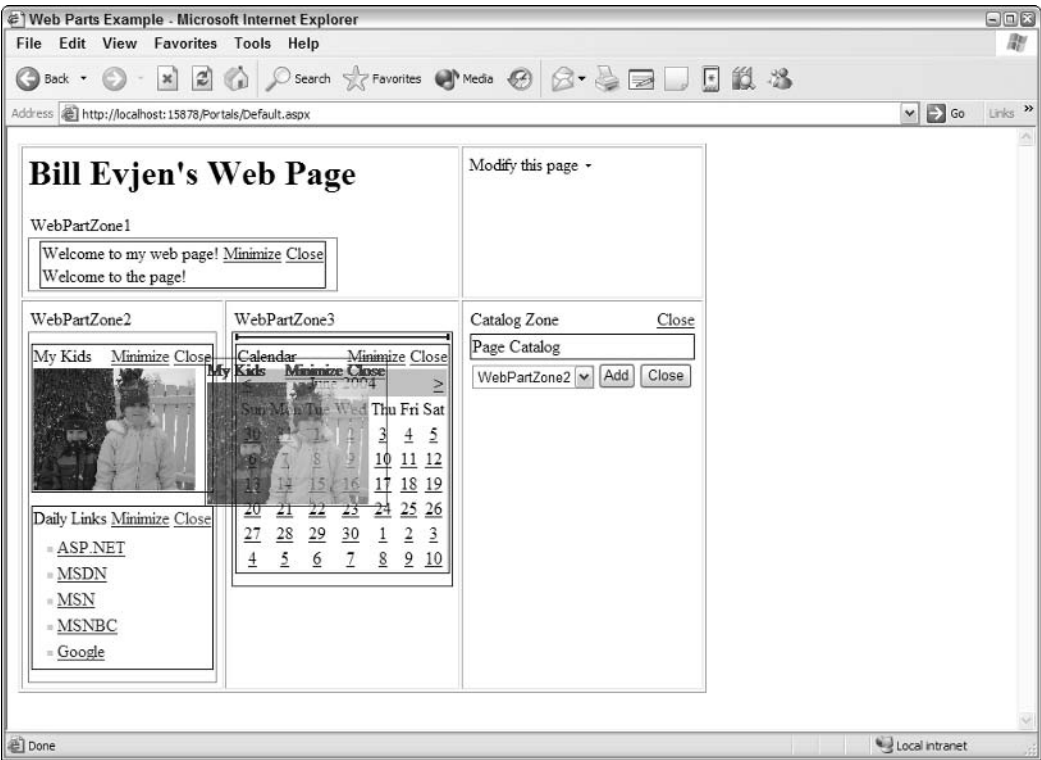


Figure 10-11

After the end user places all the items where he wants them, he can save the locations of the items for later use. When he reopens the browser, everything is drawn in the last state in which he left the page. This is done on a per-user basis, so any other users browsing to the same page see either their own modified results or the default view if it is a first visit to the page.

The user can then leave the Design view by opening the list of options from the WebPartPageMenu control and selecting Browse this Page.

Another way to move Web Parts is to enter the catalog mode of the page (which is the first option in the list from the WebPartPageMenu control). The catalog mode enables you to add deleted items to the page, and it also allows you to modify the location of the items on the page by providing the same drag-and-drop capability as the design mode.

Modifying the Web Part Settings

The third option in the WebPartPageMenu control drop-down list is Modify Web Part Settings. This selection allows the end user to modify settings determining appearance, behavior, and layout for particular Web Parts on the page.

To make this functionality work, you must add an Editor Zone to the page just as you add the Catalog Zone. This is illustrated in Listing 10-5. You place this bit of new code within the same table directly below the Catalog Zone declaration.

Listing 10-5: Adding an Editor Zone to the page

```

<td>
  <asp:CatalogZone ID="Catalogzone1" Runat="server">
    <ZoneTemplate>
      <asp:PageCatalogPart ID="Pagecatalogpart1" Runat="server" />
    </ZoneTemplate>
  </asp:CatalogZone>
  <asp:EditorZone ID="Editorzone1" Runat="server">
    <ZoneTemplate>
      <asp:AppearanceEditorPart ID="Appearanceeditorpart1" Runat="server" />
      <asp:BehaviorEditorPart ID="Behavioreditorpart1" Runat="server" />
      <asp:LayoutEditorPart ID="Layouteditorpart1" Runat="server" />
    </ZoneTemplate>
  </asp:EditorZone>
</td>

```

Just like the `<asp:CatalogZone>`, the `<asp:EditorZone>` control is a templated control that requires a `<ZoneTemplate>` section. Within this section, you can place controls that allow for the modification of the appearance, behavior, and layout of the selected Web Part. These controls include the `<asp:AppearanceEditorPart>`, `<asp:BehaviorEditorPart>`, and `<asp:LayoutEditorPart>` controls.

When you run this new section of code and select **Modify Web Part Settings** from the `WebPartPageMenu` control, you cause an arrow to appear next to the Web Part title. Clicking this arrow shows an **Edit** option, as illustrated in Figure 10-12.



Figure 10-12

After you select the **Edit** option, the right column of the table shows the various editing sections for this particular Web Part.

The **Appearance** section enables the end user to change the Web Part's details, including the title, how the title appears, and other appearance-related items such as the item's height and width. The **Appearance** section is shown in Figure 10-13.

Appearance

Title:

My Kids

Height:

Width:

Chrome Type:

Default

Hidden:

☐

Direction:

Not Set

Figure 10-13

The Behavior section (shown in Figure 10-14) enables the end user to select whether the Web Part can be closed, minimized, or exported. It also gives the end user the capability to change the roles of users who can view the Behavior section. The Behavior section is generally used if you want to allow site editors (or admins) to change the dynamics of how end users can modify Web Parts. General viewers of the page most likely won't see this section.

Behavior

Allow Close:

☐

Allow Hide:

☐

Allow Minimize:

☐

Allow Zone Change:

☐

Export Mode:

Do not allow

Help Mode:

Modal

Description:

Title Link:

Title Icon Image Link:

Catalog Icon Image Link:

Help Link:

Import Error Message:

Allow Edit:

☐

Figure 10-14

The Layout section (shown in Figure 10-15) enables the end user to change the order in which Web Parts appear in a zone or allows the end user to move Web Parts from one zone to another. This is quite similar to the drag-and-drop capabilities illustrated previously, but this section allows for the same capabilities through the manipulation of simple form elements.

Figure 10-15

After the appearance or the layout of the Web Parts is to your liking, simply click OK or Apply.

Connecting Web Parts

The last option in the WebPartPageMenu control is Connect Web Parts on this Page. This section enables you to make property connections between two Web Parts on the same page. For example, within the Weather Web Part built into one of ASP.NET's pre-built applications, you can have a separate Web Part that is simply a text box and a button that allows the end user to input a zip code. This, in turn, modifies the contents in the original Weather Web Part.

Customizing the look and feel of the WebPartPageMenu control

You can completely customize the look and feel of all the items used throughout the framework, just as you can modify all aspects of the Portal Framework. Because so many options are available to developers, I cannot possibly cover every point of customization. To see what I mean, take a look at a modified WebPartPageMenu control in particular, as shown in Listing 10-6.

Listing 10-6: Modifying the look and feel of the WebPartPageMenu control

```
<asp:WebPartPageMenu ID="Webpartpagemenu1" Runat="server" Text="MODIFY PAGE"
  Font-Size="8" Font-Names="Verdana" MenuStyle-GridLines="Horizontal"
  MenuStyle-Font-Size="8" MenuStyle-Font-Names="Verdana"
  MenuStyle-BorderColor="Gray" MenuStyle-BorderStyle="Solid"
  MenuStyle-BorderWidth="1" HoverStyle-BackColor="Gainsboro"
  VerbHoverStyle-BackColor="Khaki">
</asp:WebPartPageMenu>
```

A huge list of possible modifications can be made to this control — I selected just a few of the possible choices! As you can see, I changed the text so that it is capitalized and displayed in a different font. I also designed the background color of the text to change if the end user hovers the mouse over the text. After the end user opens the WebPartPageMenu control, the box in which the items are displayed changes as well. Notice that an option changes to a different color when the end user hovers the mouse over that particular option. Running this example produces a WebPartPageMenu like the one shown in Figure 10-16.

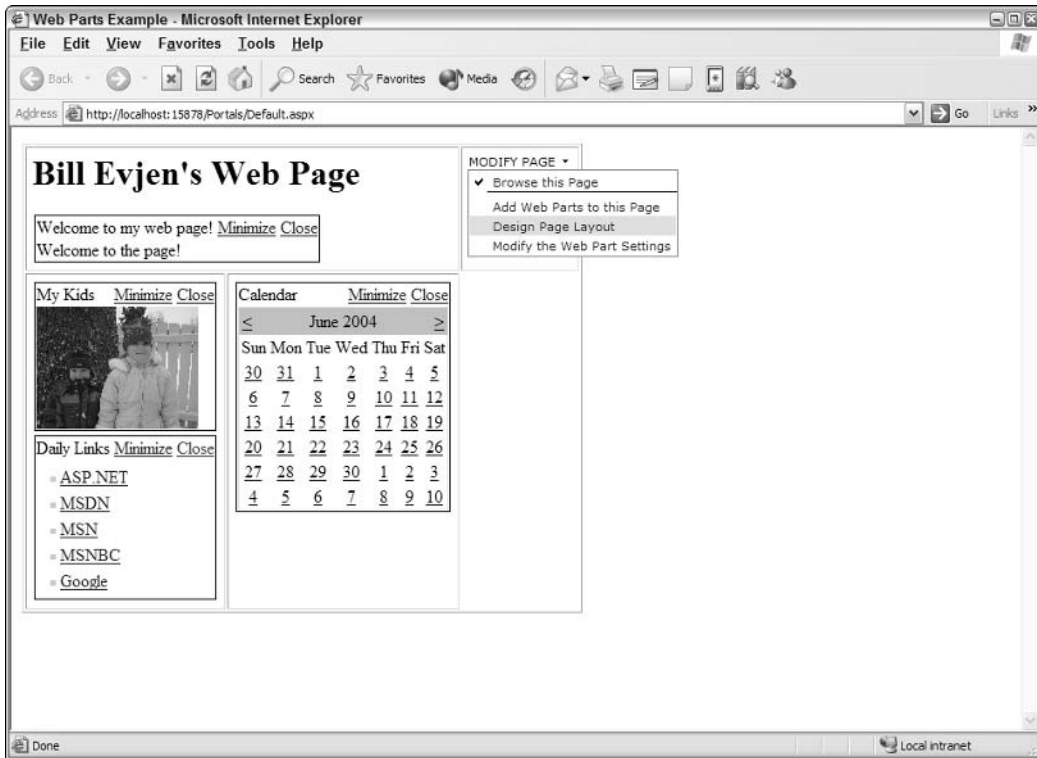


Figure 10-16

Modifying zones

One aspect of the Portal Framework that merits special attention is the capability to modify zones on the page. These zones allow for a high degree of modification — not only in the look and feel of the items placed in the zone, but also in terms of the behaviors of zones and the items contained in the zones as well. Following are some examples of what you can do to modify zones.

Turning off the capability for modifications in a zone

As you have seen, giving end users the capability to move Web Parts around the page is quite easy, whether within a zone or among entirely different zones. When working with the Portal Framework and multiple zones on a page, you do not always want to allow the end user to freely change the items that appear in every zone. You want the items placed in some zones to be left alone. Listing 10-7 shows an example of this.

Listing 10-7: Turning off the zone modification capability

```
<asp:WebPartZone ID="WebPartZone1" Runat="server"
  LayoutOrientation="Horizontal" AllowLayoutChange="false">
  <ZoneTemplate>
```

```
<asp:Label ID="Label1" Runat="server" Text="Label"
    Title="Welcome to my web page!">
    Welcome to the page!
</asp:Label>
</ZoneTemplate>
</asp:WebPartZone>
```

In this example, the first Web Part Zone, `WebPartZone1`, uses the `AllowLayoutChange` attribute with a value of `False`, which turns off the end user's capability to modify this particular Web Part Zone. When you run this page and go to the design mode, notice that you cannot drag and drop any of the Web Parts from the other zones into `WebPartZone1`. Neither can you grab hold of the Label Web Part contained in `WebPartZone1`. No capability exists to minimize and close the Web Parts contained in this zone. It allows absolutely no modifications.

Here is another interesting change you may notice when you are working in the page catalog mode with the `AllowLayoutChange` attribute set to `False`. After you select items to add to the page through the page catalog, `WebPartZone1` does not appear in the drop-down list of places where you can publish the Web Parts (see Figure 10-17). From this figure, you can see that only `WebPartZone2` and `WebPartZone3` appear and allow modifications.

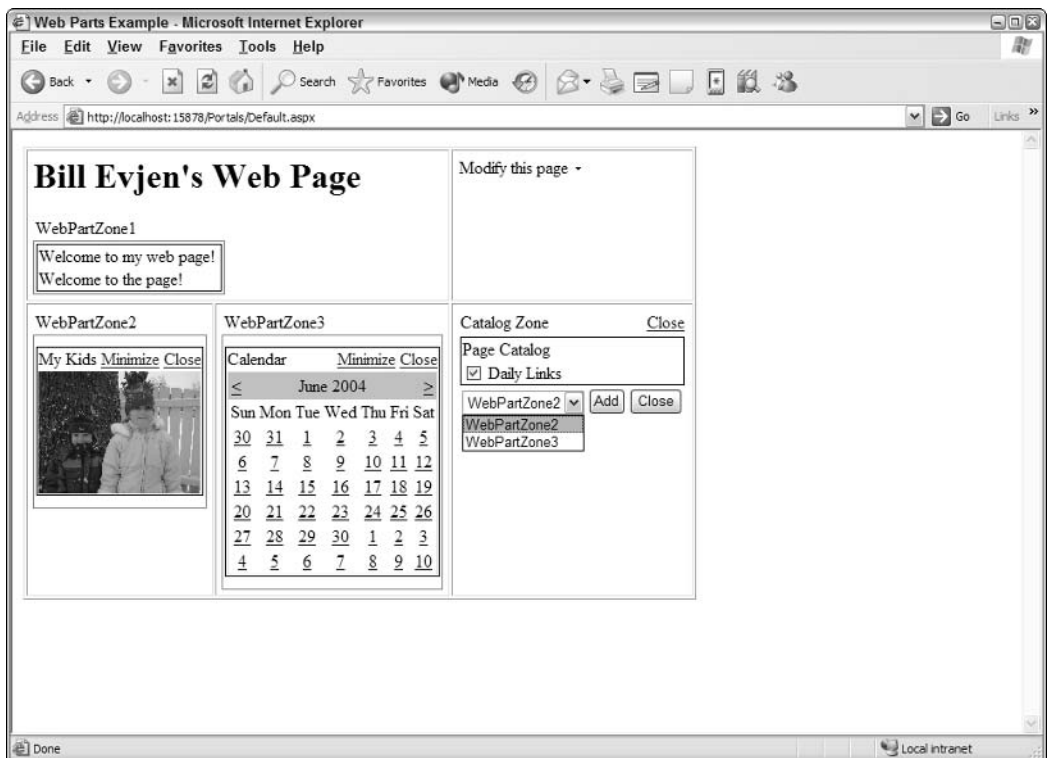


Figure 10-17

Adding controls through other means

Earlier in this chapter, you examined how to use the `<asp:PageCatalogPart>` control to restore controls to a page after they had been deleted. Although the `<asp:PageCatalogPart>` is ideal for this, you might also want to allow the end user to add Web Parts that are not on the page by default. You may want to enable the end user to add more than one of any particular Web Part to a page. For these situations, you work with the `<asp:DeclarativeCatalogPart>` control.

Listing 10-8 shows an example of using this type of catalog system in place of the `<asp:PageCatalogPart>` control.

Listing 10-8: Using the `DeclarativeCatalogPart` control

```
<asp:CatalogZone ID="Catalogzone1" Runat="server">
  <ZoneTemplate>
    <asp:DeclarativeCatalogPart ID="Declarativecatalogpart1" Runat="server">
      <WebPartsTemplate>
        <uc1:CompanyContactInfo ID="CompanyContact" Runat="Server" />
        <uc1:PhotoAlbum ID="PhotoAlbum" Runat="Server" />
        <uc1:Customers ID="Customers" Runat="Server" />
        <uc1:Locations ID="Locations" Runat="Server" />
      </WebPartsTemplate>
    </asp:DeclarativeCatalogPart>
  </ZoneTemplate>
</asp:CatalogZone>
```

Instead of using the `<asp:PageCatalogPart>` control, this catalog uses the `<asp:DeclarativeCatalogPart>` control. This templated control needs a `<WebPartsTemplate>` section where you place all the available controls. The controls appear in the check box list in the same order in which you declare them in the `<WebPartsTemplate>` section. Figure 10-18 shows how the catalog will look in the Design view in Visual Studio 2005.

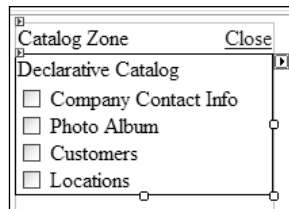


Figure 10-18

This catalog lets you select items from the list of Web Parts and assign the location of the zone in which they will be placed. Once placed, notice that the option to add these Web Parts has not disappeared as it did with the earlier `PageCatalogPart` control. In fact, you can add as many of these items to the page as you deem necessary — even if it is to the same zone within the Portal Framework.

Even using the `DeclarativeCatalogPart` control is not always 100% ideal. When the end user closes one of the Web Parts that initially appears on the page, he does not see that control listed in the `DeclarativeCatalogPart` control's list of elements. In fact, the end user cannot re-add these deleted items. Using both the

PageCatalogPart control and the DeclarativeCatalogPart control simultaneously is the best solution. The great thing about this framework is that it allows you to do that. The Portal Framework melds both controls into a cohesive control that not only enables you to add controls that are not on the page by default, but it also lets you add previously deleted default controls. Listing 10-9 shows an example of this.

Listing 10-9: Combining both catalog types

```
<asp:CatalogZone ID="Catalogzone1" Runat="server">
  <ZoneTemplate>
    <asp:PageCatalogPart ID="Pagecatalogpart1" Runat="server" />
    <asp:DeclarativeCatalogPart ID="Declarativecatalogpart1" Runat="server">
      <WebPartsTemplate>
        <uc1:CompanyContactInfo ID="CompanyContact" Runat="Server" />
        <uc1:PhotoAlbum ID="PhotoAlbum" Runat="Server" />
        <uc1:Customers ID="Customers" Runat="Server" />
        <uc1:Locations ID="Locations" Runat="Server" />
      </WebPartsTemplate>
    </asp:DeclarativeCatalogPart>
  </ZoneTemplate>
</asp:CatalogZone>
```

In this example, both the PageCatalogPart control and the DeclarativeCatalogPart control are contained with the <ZoneTemplate> section. When this page is run, you see the results shown in Figure 10-19.

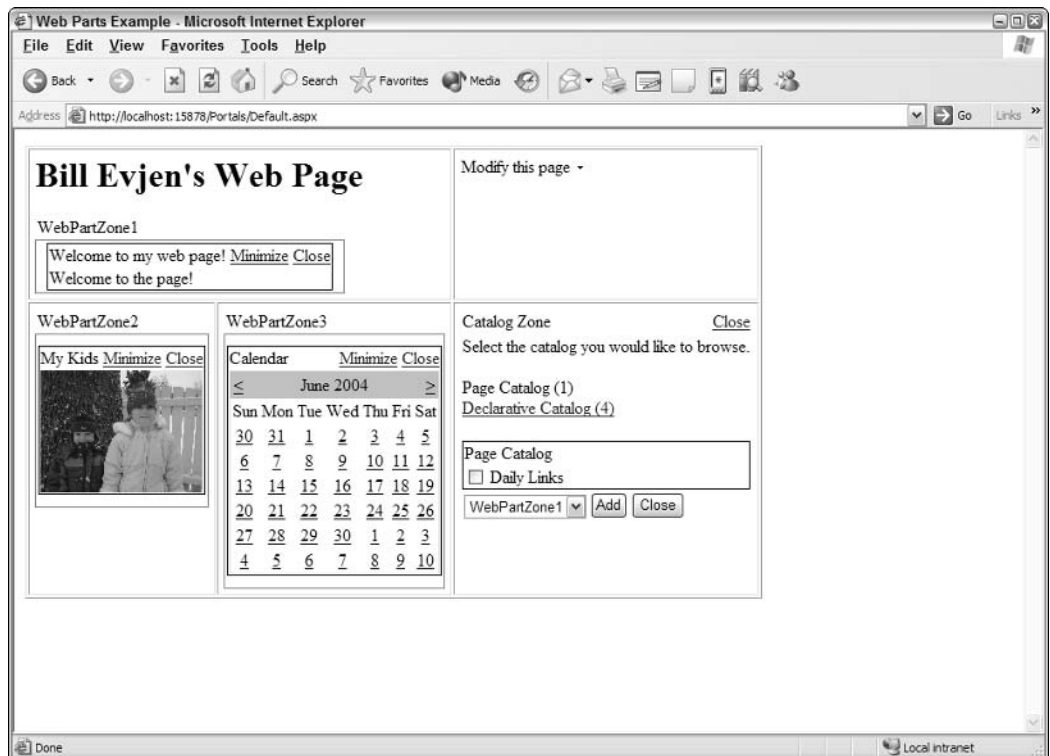


Figure 10-19

Chapter 10

You can see that each catalog is defined within the Catalog Zone. Figure 10-19 shows the PageCatalogPart control's collection of Web Parts (defined as Page Catalog), while a link to the Declarative Catalog is provided for that particular list of items. Note that the order in which the catalogs appear in the `<ZoneTemplate>` section is the order in which the links appear in the Catalog Zone.

Web Part verbs

Web Part verbs declare the actions of the items (such as Minimize and Close) that appear in the title. These verbs are basically links that initiate an action for a particular Web Part. The available list of Web Part verbs includes

- ☐ `<CloseVerb>`
- ☐ `<ConnectVerb>`
- ☐ `<EditVerb>`
- ☐ `<ExportVerb>`
- ☐ `<HelpVerb>`
- ☐ `<MinimizeVerb>`
- ☐ `<RestoreVerb>`

The `<asp:WebPartZone>` control allows you to control these verbs by nesting the appropriate verb elements within the `<asp:WebPartZone>` element itself. After these are in place, you can manipulate how these items appear in all the Web Parts that appear in the chosen Web Part Zone.

For example, look at graying out the default Close link included with a Web Part. This is illustrated in Listing 10-10.

Listing 10-10: Graying out the Close link in a Web Part

```
<asp:WebPartZone ID="WebPartZone3" Runat="server">
  <CloseVerb Enabled="False" />
  <ZoneTemplate>
    <asp:Calendar ID="Calendar1" Runat="server">
      </asp:Calendar>
    </ZoneTemplate>
  </asp:WebPartZone>
```

In this example, you can see that you simply need to set the `Enabled` attribute of the `<CloseVerb>` element to `False` in order to gray out the Close link in any of the generated Web Parts included in this Web Part Zone. If you construct the Web Part Zone in this manner, you achieve the results shown in Figure 10-20.

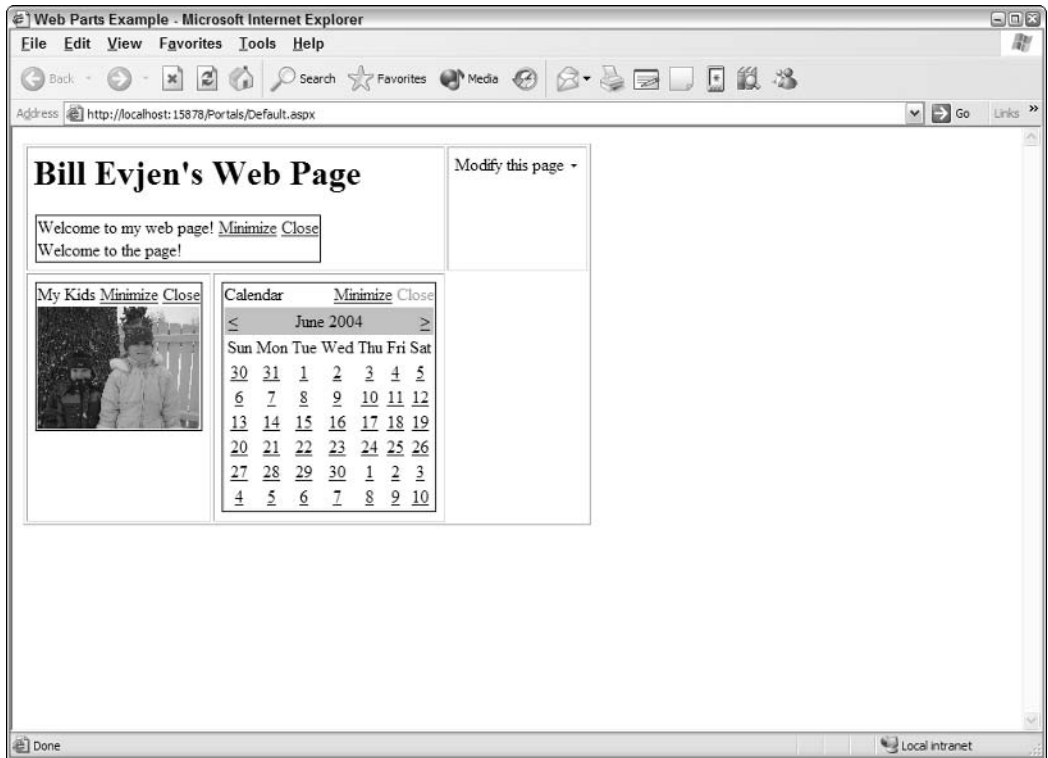


Figure 10-20

If you don't want to gray out the Close link (or any other verb link contained within the Web Part), you must instead use the `Visible` attribute of the appropriate verb (see Listing 10-11).

Listing 10-11: Removing the Close link in a Web Part

```
<asp:WebPartZone ID="WebPartZone3" Runat="server">
  <CloseVerb Visible="False" />
  <ZoneTemplate>
    <asp:Calendar ID="Calendar1" Runat="server">
      </asp:Calendar>
    </ZoneTemplate>
  </asp:WebPartZone>
```

Using the `Visible` attribute produces the screen shown in Figure 10-21.

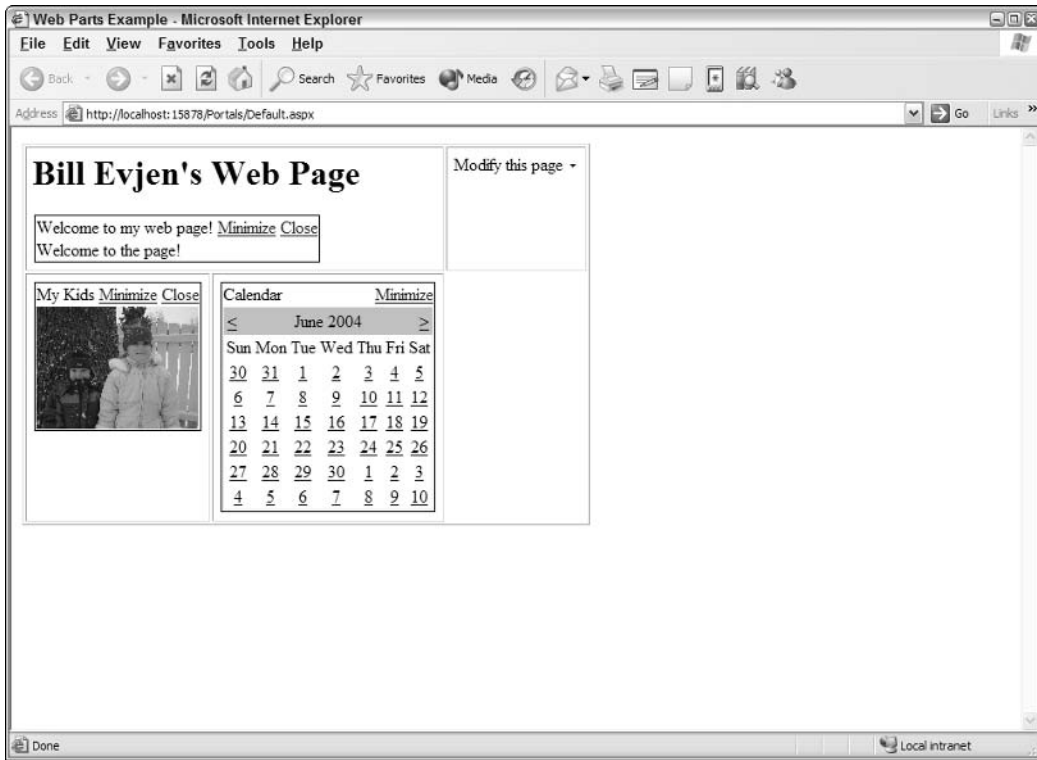


Figure 10-21

Verb elements provide another exciting feature: They give you the capability to use images for the items rather than default text. Using images instead of text makes the Web Parts appear more like the overall Windows environment. For instance, you can change the contents of `WebPartZone3` again so that it now uses images instead of text for the Close and Minimize links. This is illustrated in Listing 10-12.

Listing 10-12: Using images for the Web Part verbs

```
<asp:WebPartZone ID="WebPartZone3" Runat="server">
  <CloseVerb ImageUrl="Images/CloseVerb.gif" />
  <MinimizeVerb ImageUrl="Images/MinimizeVerb.gif" />
  <ZoneTemplate>
    <asp:Calendar ID="Calendar1" Runat="server">
      </asp:Calendar>
    </ZoneTemplate>
  </asp:WebPartZone>
```

To point to an image for the verb, use the `ImageUrl` attribute. This produces something similar to Figure 10-22, depending on the images you use.

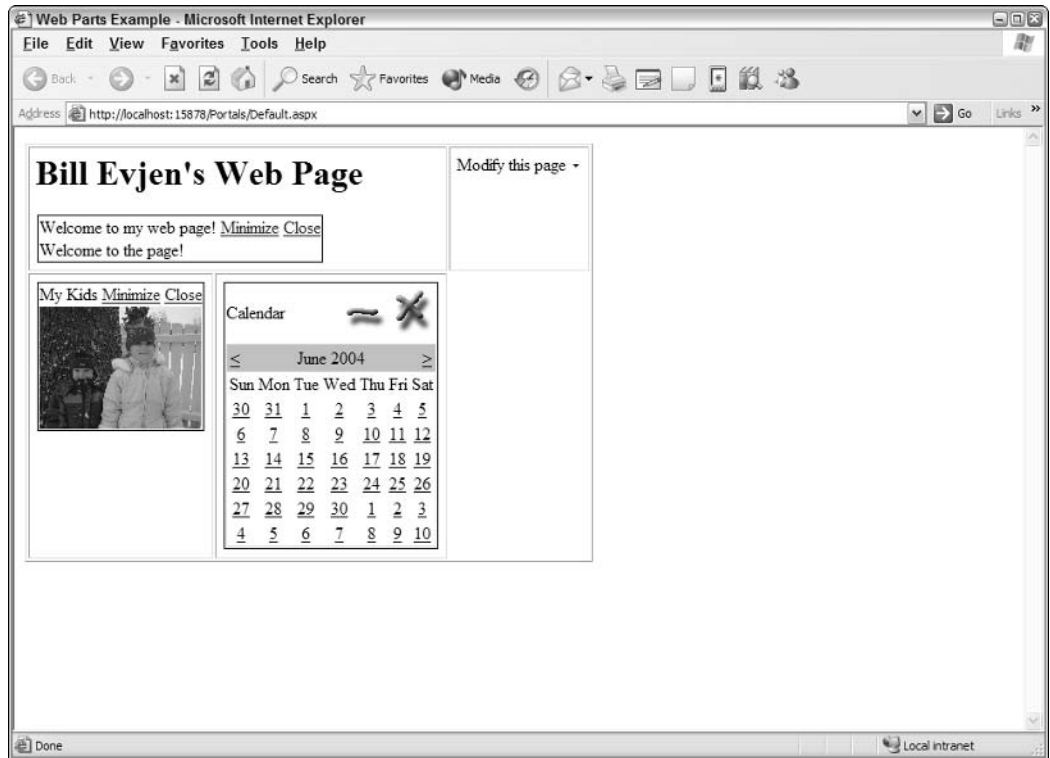


Figure 10-22

This chapter thus far has concentrated on creating completely customizable portal applications in a declarative manner using the capabilities provided by the ASP.NET Portal Framework. As with most aspects of ASP.NET, however, not only can you work with appearance and functionality in a declarative fashion, but you can also create the same constructs through server-side code.

Working with Classes in the Portal Framework

The Portal Framework provides three main classes for dealing with the underlying framework presented in this chapter: `WebPartManager`, `WebPartZone`, and `WebPart`.

The `WebPartManager` class allows you to perform multiple operations in your server-side code. The following table shows a partial listing of some of the properties that this class provides.

WebPartManager Class Properties	Description
Connections	Provides a collection of all the connections between Web Parts contained on the page.
DisplayMode	Allows you to change the page's display mode. Possible choices include <code>CatalogDisplayMode</code> , <code>ConnectDisplayMode</code> , <code>DesignDisplayMode</code> , <code>EditDisplayMode</code> , and <code>NormalDisplayMode</code> .
EnableCatalogDisplayMode	Takes a <code>Boolean</code> value and either enables or disables the capability to turn on the page in the catalog mode.
EnableConnectDisplayMode	Takes a <code>Boolean</code> value and either enables or disables the capability to turn on the page in the connect mode.
EnableDesignDisplayMode	Takes a <code>Boolean</code> value and either enables or disables the capability to turn on the page in the design mode.
EnableEditDisplayMode	Takes a <code>Boolean</code> value and either enables or disables the capability to turn on the page in the display mode.
SelectedWebPart	Allows you to perform multiple operations on the selected Web Part.
WebParts	Provides a collection of all the Web Parts contained on the page.
Zones	Provides a collection of all the Web Part Zones contained on the page.

Beyond the properties of the `WebPartManager` class, you also have an extensive list of available methods at your disposal. The following table outlines some of the available methods of the `WebPartManager` class.

WebPartManager Class Methods	Description
AddWebPart	Allows you to dynamically add new Web Parts to a particular zone on the page.
ConnectWebParts	Allows you to connect two Web Parts together via a common property or value.
DeleteWebPart	Allows you to dynamically delete new Web Parts from a particular zone on the page.
DisconnectWebParts	Allows you to delete a connection between two Web Parts.
MoveWebPart	Allows you to move a Web Part from one zone to another, or allows you to change the index order in which Web Parts appear in a particular zone.

Whereas the `WebPartManager` class allows you to manipulate the location, addition, and deletion of Web Parts that appear in the page as a whole, the `WebPartZone` class allows you to modify a single Web Part Zone on the page. The following table provides a list of some properties available to the `WebPartZone` class.

WebPartZone Class Properties	Description
<code>AllowLayoutChange</code>	Takes a <code>Boolean</code> value and either enables or disables the Web Part Zone's capability to accept or allow any changes in the Web Parts it contains.
<code>BackColor</code> , <code>BackImageUrl</code> , <code>BorderColor</code> , <code>BorderStyle</code> , <code>BorderWidth</code>	Enable you to modify the Web Part Zone's general appearance.
<code>CloseVerb</code>	References the Close verb for a particular Web Part Zone from which you can then manipulate the verb's <code>Description</code> , <code>Enabled</code> , <code>ImageUrl</code> , <code>Text</code> , and <code>Visible</code> properties.
<code>ConnectVerb</code>	References a Web Part Zone's Connect verb from which you can then manipulate the verb's <code>Description</code> , <code>Enabled</code> , <code>ImageUrl</code> , <code>Text</code> , and <code>Visible</code> properties.
<code>DragHighlightColor</code>	Takes a <code>System.Color</code> value that sets the color of the Web Part Zone's border if focused when the moving of Web Parts is in operation. This also changes the color of the line that appears in the Web Part Zone specifying where to drop the Web Part.
<code>EditVerb</code>	References a Web Part Zone's Edit verb from which you can then manipulate the verb's <code>Description</code> , <code>Enabled</code> , <code>ImageUrl</code> , <code>Text</code> , and <code>Visible</code> properties.
<code>EmptyZoneText</code>	Sets the text that is shown in the zone if a Web Part is not set in the zone.
<code>HeaderAlignment</code>	Allows you to align the Web Part Zone header.
<code>HeaderText</code>	Sets header text.
<code>Height</code>	Sets the height of the Web Part Zone.
<code>HelpVerb</code>	References a Web Part Zone's Help verb from which you can then manipulate the verb's <code>Description</code> , <code>Enabled</code> , <code>ImageUrl</code> , <code>Text</code> , and <code>Visible</code> properties.
<code>MenuImageUrl</code> , <code>MenuLabelStyle</code> , <code>MenuLabelText</code> , <code>MenuText</code>	Enable you to modify the drop-down menu that appears when end users edit a Web Part. These properties let you apply an image, alter the text, or change the style of the menu.
<code>MinimizeVerb</code>	References a Web Part Zone's Minimize verb from which you can then manipulate the verb's <code>Description</code> , <code>Enabled</code> , <code>ImageUrl</code> , <code>Text</code> , and <code>Visible</code> properties.

Table continued on following page

WebPartZone Class Properties	Description
Orientation	Enables you to change the Web Part Zone's orientation from horizontal to vertical or vice versa.
RestoreVerb	References a Web Part Zone's Restore verb, from which you can then manipulate the verb's Description, Enabled, ImageUrl, Text, and Visible properties.
VerbButtonType	Enables you to change the button style. Choices include <code>ButtonType.Button</code> , <code>ButtonType.Image</code> , or <code>ButtonType.Link</code> .
WebParts	Provides a collection of all the Web Parts contained within the zone.
Width	Sets the width of the Web Part Zone.

You have a plethora of options to manipulate the look and feel of the Web Part Zone and the items contained therein.

The final class is the `WebPart` class. This class enables you to manipulate specific Web Parts located on the page. The following table details some of the available properties of the `WebPart` class.

WebPart Class Properties	Description
AllowClose	Takes a <code>Boolean</code> value that specifies whether the Web Part can be closed and removed from the page.
AllowEdit	Takes a <code>Boolean</code> value that specifies whether the end user can edit the Web Part.
AllowHide	Takes a <code>Boolean</code> value that specifies whether the end user can hide the Web Part within the Web Part Zone. If the control is hidden, it is still in the zone, but invisible.
AllowMinimize	Takes a <code>Boolean</code> value that specifies whether the end user can collapse the Web Part.
AllowPaginate	Takes a <code>Boolean</code> value that specifies whether the Web Part can be paginated using the ASP.NET Pager server control.
AllowZoneChange	Takes a <code>Boolean</code> value that specifies whether the end user can move the Web Part from one zone to another.
BackColor, BackImageUrl, BorderColor, BorderStyle, BorderWidth	Enable you to modify the Web Part's general appearance.
Caption	Sets an item of text directly in the title bar next to the Web Part's title. This property allows you to differentiate among multiple Web Parts that have the same title.

WebPart Class Properties	Description
ChromeState	Specifies whether the Web Part chrome is in a normal state or is minimized.
ChromeType	Specifies the chrome type that the Web Part uses. Available options include <code>BorderOnly</code> , <code>Default</code> , <code>None</code> , <code>TitleAndBorder</code> , and <code>TitleOnly</code> .
Direction	Specifies the direction of the text or items placed within the Web Part. Available options include <code>LeftToRight</code> , <code>NotSet</code> , and <code>RightToLeft</code> . This property is ideal for dealing with Web Parts that contain Asian text that is read from right to left.
HelpMode	Specifies how the help items display when the end user clicks on the Help verb. Available options include <code>Modal</code> , <code>Modeless</code> , and <code>Navigate</code> . <code>Modal</code> displays the help items within a modal window if the end user's browser supports modal windows. If not, a pop-up window displays. <code>Modeless</code> means that a pop-up window displays for every user. <code>Navigate</code> redirects the user to the appropriate help page (specified by the <code>HelpUrl</code> property) when he clicks on the Help verb.
HelpUrl	Used when the <code>HelpMode</code> is set to <code>Navigate</code> . Takes a <code>String</code> value that specifies the location of the page the end user is redirected to when he clicks on the Help verb.
ScrollBars	Applies scroll bars to the Web Part. Available values include <code>Auto</code> , <code>Both</code> , <code>Horizontal</code> , <code>None</code> , and <code>Vertical</code> .
Title	Specifies the text for the Web Part's title. Text appears in the title bar section.
TitleIconImageUrl	Enables you to apply an icon to appear next to the title by specifying to the icon image's location as a <code>String</code> value of the property.
TitleUrl	Specifies the location to direct the end user when the Web Part's title Web Part is clicked. When set, the title is converted to a link; when not set, the title appears as regular text.
Zone	Allows you to refer to the zone in which the Web Part is located.

Summary

This chapter introduced you to the Web Part Manager, Web Part Zone, and the Web Part controls. Not only do these controls allow for easy customization of the look and feel of either the Web Parts or the zones in which they are located, but you can also use the framework provided to completely modify the behavior of these items.

Personally, I find the Portal Framework to be one of the more exciting new features of ASP.NET 2.0. I like the idea of creating completely modular and customizable Web pages. End users like this feature, and it is quite easy for developers to implement. Just remember that you don't have to implement every feature explained in this chapter, but with the framework provided, you can choose only the functionality that you want.

11

SQL Cache Invalidation

Performance is key for any application or piece of code that you develop. The browser helps with client-side caching of text and images, whereas the server-side caching you choose to implement is vital for creating the best possible performance. Caching is the process of storing frequently used data on the server to fulfill subsequent requests. You will discover that grabbing objects from memory is much faster than re-creating the Web pages or items contained in them from scratch each and every time. Caching increases your application's performance, scalability, and availability. The more you fine-tune your application's caching approach, the better it performs.

ASP.NET 2.0 introduces SQL Server cache invalidation that enables you to create a cache based upon items contained within a Microsoft SQL Server database and also to invalidate it if any changes occur in the database. This is something frequently requested by developers using ASP.NET 1.0/1.1, and the ASP.NET team worked hard to bring it to ASP.NET 2.0. This chapter takes a close look at this unique aspect of caching.

Caching in ASP.NET 1.0/1.1

In ASP.NET 1.0/1.1, developers deal with caching in several ways. First, you can cache an entire HTTP response (the entire Web page) using a mechanism called output caching. Two other methods are partial page caching and data caching. These methods are described in the following sections.

Output caching

Output caching is a way to keep the dynamically generated page content in the server's memory for later retrieval. After a cache is saved to memory, it can be used when any subsequent requests are made to the server. You apply output caching using the `OutputCache` directive as follows:

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

You apply output caching by inserting an `OutputCache` page directive at the top of an `.aspx` page. The `Duration` attribute defines the number of seconds that a page is stored in memory. The `VaryByParam` attribute determines which versions of the page output are actually cached. You can generate different responses based on whether an HTTP-POST or HTTP-GET response is required. Other than the attributes for the `OutputCache` directive, ASP.NET 1.0/1.1 includes the `VaryByHeader`, `VaryByCustom`, and `Location` attributes.

Partial page caching

Similar to output caching, partial page caching enables you to cache only specific blocks of a Web page. You can, for example, cache only the center of the page. Partial page caching is achieved with the caching of user controls. You can build your ASP.NET pages utilizing numerous user controls and then apply output caching to the user controls that you select. This, in essence, only caches the parts of the page that you want, while leaving other parts of the page outside the reach of caching. This is a nice feature, and if done correctly, it can lead to pages that perform better.

Data caching using the Cache object

The other method of caching is using the `Cache` object to start caching specific data items for later use on a particular page or group of pages. The `Cache` object enables you to store everything from simple name/value pairs to more complex objects like datasets and even entire `.aspx` pages.

The `Cache` object is used in the following fashion:

```
Cache("WhatINeedToStore") = myDataSet
```

After an item is in the cache, you can retrieve it later as shown here:

```
Dim ds As New DataSet  
ds = CType(Cache("WhatINeedToStore"), DataSet)
```

The `Cache` object is an outstanding way to cache your pages and is, in fact, what the `OutputCache` directive uses under the covers.

Cache dependencies

Using the `Cache` object, you can store items in the cache and invalidate these items in the cache based on several different dependencies. In ASP.NET 1.0/1.1, the only possible dependencies include

- ☐ File-based dependencies
- ☐ Key-based dependencies
- ☐ Time-based dependencies

When inserting items into the cache using the `Cache` object, you set the dependencies with the `Insert` method, as shown in the following example:

```
Cache.Insert("DSN", connectionString, _  
    New CacheDependency(Server.MapPath("myconfig.xml")))
```

By using a *dependency*, when the item being referenced changes, the cache for that item is removed from memory.

ASP.NET 2.0 unseals the *CacheDependency* class

The big change in ASP.NET 2.0 in the area of caching is that the *CacheDependency* class has been unsealed (or made overrideable). You can now create classes that inherit from the *CacheDependency* class and create more elaborate dependencies that are not limited to the *Time*, *Key*, or *File* dependencies of the past.

When you create your own cache dependencies, you have the option to add procedures for such things as Web services data, only-at-midnight dependencies, or textual string changes within a file. The dependencies you create are limited only by your imagination. The unsealing of the *CacheDependency* class has provided all this for you.

Because of the unsealing of the *CacheDependency* class, the ASP.NET team has built a new SQL Server cache dependency — *SqlCacheDependency*. A SQL cache dependency was the most requested caching feature from ASP.NET 1.0/1.1 developers. You can now determine that a cache becomes invalid whenever a table changes within the underlying SQL Server.

Using the SQL Server Cache Dependency

To utilize the new SQL Server cache dependency feature in ASP.NET 2.0, you must perform a one-time setup of your SQL Server database. To set up your SQL Server, use the *aspnet_regsqlcache.exe* tool found at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxxx`. This tool makes the necessary modifications to SQL Server so that you can start working with the new SQL cache invalidation features.

To get at the *aspnet_regsqlcache.exe* tool, open up the Visual Studio Command Prompt by choosing **Start** ⇨ **All Programs** ⇨ **Microsoft Visual Studio 2005** ⇨ **Visual Studio Tools** ⇨ **Visual Studio Command Prompt**. After you get the command prompt, type this command:

```
aspnet_regsqlcache.exe -?
```

This pulls up the help command list (see Figure 11-1).

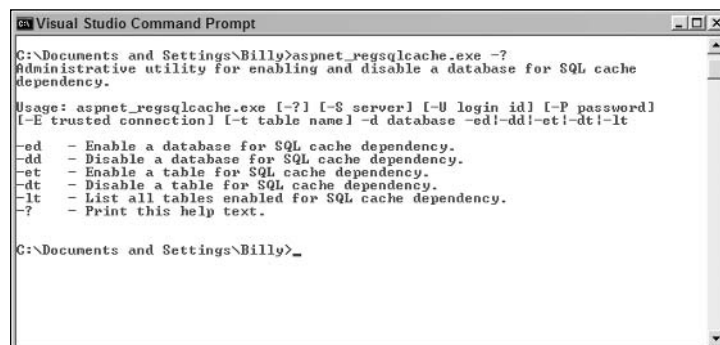


Figure 11-1

The following table details the commands available in the `aspnet_regsqlcache.exe` tool.

Command	Description
-S server	The server where the SQL cache invalidation setup is to occur.
-U login id	The SQL Server login ID.
-P password	The SQL Server password.
-E trusted connection	Instructions for using the current Windows credentials for authentication.
-t table name	The name of the table you want to work with for SQL Server cache invalidation.
-d database	The name of the database.
-ed	A command to enable a database for SQL cache dependency.
-dd	A command to disable a database for SQL cache dependency.
-et	A command to enable a table for SQL cache dependency.
-dt	A command to disable a table for SQL cache dependency.
-lt	A list of all the tables enabled for SQL cache dependency.
-?	A list of available option commands.

The following sections show you how to use some of these commands.

Enabling databases for SQL Server cache invalidation

To use SQL Server cache invalidation, begin with two steps. The first step enables the appropriate database. Next, you enable the tables that you want to work with. You must perform both steps for this process to work. Start by enabling the database.

If you want to enable your databases for SQL cache invalidation and you are working on the computer where the SQL Server instance is located, you use the following construct:

```
aspnet_regsqlcache.exe -S localhost -U sa -P password -d Northwind -ed
```

This produces something similar to what you see in Figure 11-2.

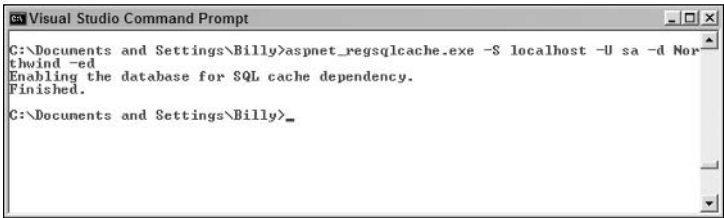


Figure 11-2

From this command prompt, you can see that I simply enabled the Northwind database (the sample database that comes with SQL Server) for SQL cache invalidation. The `aspnet_sqlcache.exe` tool responded as follows:

```
Enabling the database for SQL cache dependency.  
Finished.
```

Now that you have enabled the database for SQL cache invalidation, you can enable one or more tables contained within the Northwind database.

Enabling tables for SQL Server cache invalidation

You enable one or more tables by using the following command:

```
aspnet_regsqlcache.exe -S localhost -U sa -P password -d Northwind -t Customers -et
```

You can see that this command is not much different from the one for enabling the database, except for the extra `-t Customers` entry. `Customers` is the name of the table that is enabled, as shown in Figure 11-3.

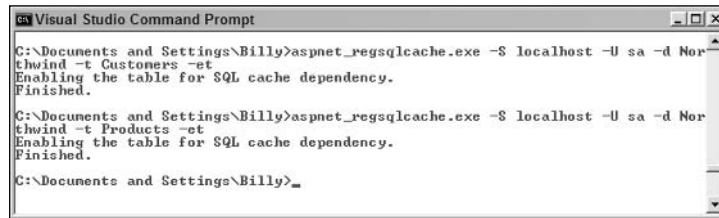


Figure 11-3

In this screen shot, you can see that I enabled two tables within the Northwind database. The first is the `Customers` table and the second is the `Products` table. After a table is successfully enabled, you receive the following response:

```
Enabling the table for SQL cache dependency.  
Finished.
```

After the table is enabled, you can begin using the SQL cache invalidation features. But before you do, the following section shows you what happens to SQL Server when you enable these features.

Looking at SQL Server

Now that both the Northwind database and the `Customers` and `Products` tables have been enabled for SQL cache invalidation, look at what happens in SQL Server to enable this. If you open up the SQL Server Enterprise Manager, you see a new table contained within the Northwind database — `AspNet_SqlCacheTablesForChangeNotification` (whew, that's a long one!). You should see what is shown in Figure 11-4.

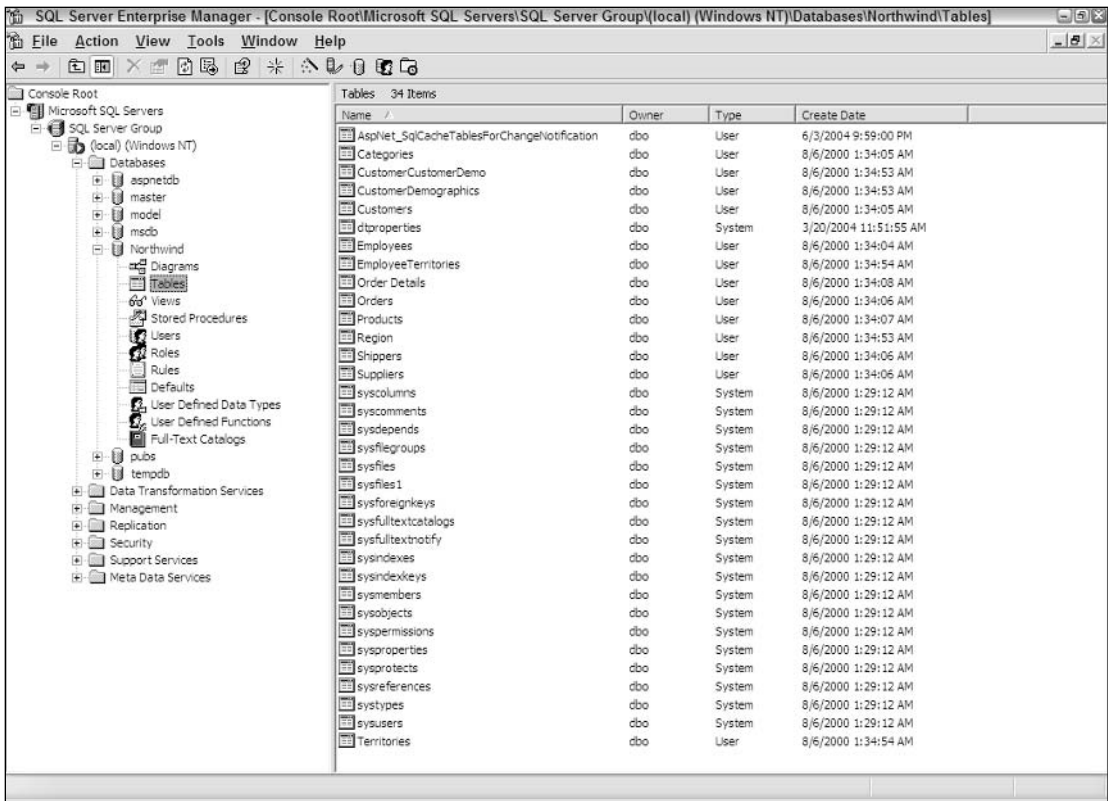


Figure 11-4

At the top of the list of tables in the right-hand pane, you see the `AspNet_SqlCacheTablesForChangeNotification` table. This is the table that ASP.NET uses to learn which tables are being monitored for change notification and also to make note of any changes to the tables being monitored. The table is actually quite simple when you look at the details, as illustrated in Figure 11-5.

tableName	notificationCreated	changeId
Customers	6/3/2004 10:03:13 PM	0
Products	6/3/2004 10:03:27 PM	0

Figure 11-5

From this screen shot, you can see three columns in this new table. The first is the `tableName` column. This column is simply a `String` reference to the name of the table contained in the same database. Any table named here is enabled for SQL cache invalidation. Therefore, Figure 11-5 shows that the `Customers` and `Products` table are enabled.

The second column, `notificationCreated`, shows the date and time when the table was enabled for SQL cache invalidation. The final column, `changeId`, is used to communicate to ASP.NET any changes to the included tables. ASP.NET is monitoring this column for changes and, depending on the value, either uses what is stored in memory or makes a new database query.

Looking at the tables that are enabled

Using the `aspnet_regsqlcache.exe` tool, you can see which tables are enabled in a particular database with a simple command. If you are working through the preceding examples, you see that so far you have enabled the Customers and Products tables of the Northwind database. To get a list of the tables that are enabled, use something similar to the following command:

```
aspnet_regsqlcache.exe -S localhost -U sa -P password -d Northwind -lt
```

This command produces the results shown in Figure 11-6.

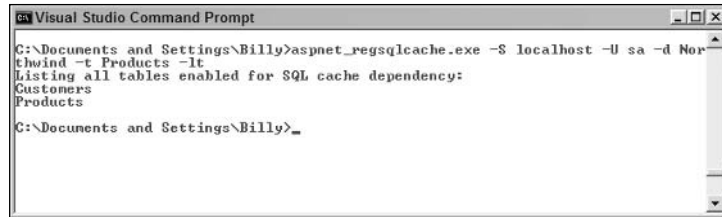


Figure 11-6

As you can see from this screen shot, the `-lt` command produces a simple list of tables enabled for SQL cache invalidation. Inputting this command produces the following results:

```

Listing all tables enabled for SQL cache dependency:
Customers
Products

```

Disabling a table for SQL Server cache invalidation

Now that you know how to enable your SQL Server database for SQL Server cache invalidation, take a look at how you remove the capability for a specific table to be monitored for this process. To remove a table from the SQL Server cache invalidation process, use the `-dt` command. Figure 11-7 details how to remove an individual table from the process.

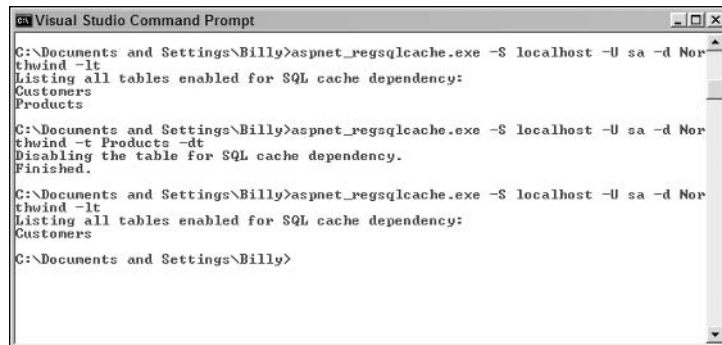


Figure 11-7

Chapter 11

From Figure 11-7, you can see that the tables enabled for SQL Server cache invalidation are listed using the `-lt` command. It shows that you have both the Customers and Products tables enabled. Next, you remove the Products table from the process using the following command:

```
aspnet_regsqlcache.exe -S localhost -U sa -P password -d Northwind -t Products -dt
```

You can see that all you do is specify the name of the table using the `-t` command followed by a `-dt` command (disable table). From the figure, you can also see the third command again lists the tables that are enabled for SQL Server cache invalidation, and this time, the Products table is not listed — instead, only the Customers table is listed.

Disabling a database for SQL Server cache invalidation

Not only can you pick and choose the tables that you want to remove from the process, but you can also disable the entire database for SQL Server cache invalidation. In order to disable an entire database, you use the `-dd` command (disable database). You should note that disabling an entire database for SQL Server cache invalidation also means that every single table contained within this database is also disabled. Figure 11-8 shows the Northwind database being disabled on my computer.

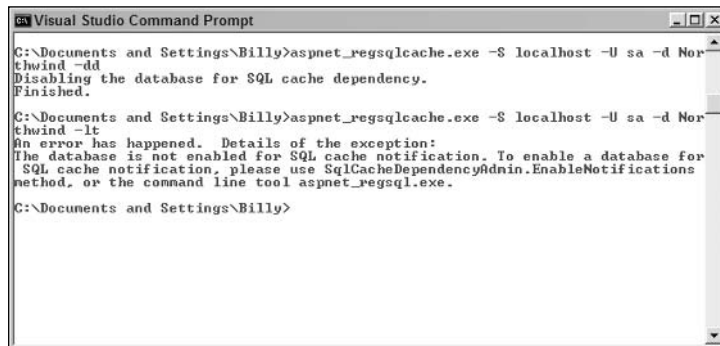


Figure 11-8

From this figure, you can see that I disabled the Northwind database from SQL Server cache invalidation using the `-dd` command. This is the result:

```
Disabling the database for SQL cache dependency.  
Finished.
```

To ensure that the table was no longer enabled for SQL Server cache invalidation, I attempted to list the tables that were enabled for cache invalidation using the `-lt` command. I received the following error:

```
An error has happened. Details of the exception:  
The database is not enabled for SQL cache notification. To enable a database for  
SQL cache notification, please use SqlCacheDependencyAdmin.EnableNotifications  
method, or the command line tool aspnet_regsql.exe.
```

If you now open the Northwind database in the SQL Server Enterprise Manager, notice that the `AspNet_SqlCacheTablesForChangeNotification` table has been removed for the database.

Configuring your ASP.NET Application

After you enable a database for SQL Server cache invalidation, and you also enable a couple of tables within this database, the next step is to configure your application for SQL Server cache invalidation.

To configure your application to work with SQL Server cache invalidation, the first step is to make some changes to the `web.config` file. In the `web.config` file, specify that you want to work with the Northwind database and you want ASP.NET connected to it.

Listing 11-1 shows an example of how you should change your `web.config` file in order to work with SQL Server cache invalidation.

Listing 11-1: Configuring the web.config file

```
<configuration>

  <connectionStrings>
    <add name="AppConnectionString1" connectionString="Provider=SQLOLEDB.1;
      Data Source=EVJEN01;User ID=sa;Password=;Initial
      Catalog=Northwind;Persist Security Info=False"
      providerName="System.Data.OleDb" />
  </connectionStrings>

  <system.web>

    <caching>
      <sqlCacheDependency enabled="true">
        <databases>
          <add name="Northwind" connectionStringName="AppConnectionString1"
            pollTime="500" />
        </databases>
      </sqlCacheDependency>
    </caching>

  </system.web>
</configuration>
```

From this listing, you can see that the first thing established is the connection string to the Northwind database using the `<connectionStrings>` element in the `web.config` file. It is important to make note of the name of the connection string because it is utilized later in the configuration settings for SQL Server cache invalidation.

The SQL Server cache invalidation is configured using the new `<caching>` element. This element must be nested within the `<system.web>` elements. Because you are working with a SQL Server cache dependency, you must use a `<sqlCacheDependency>` child node, and you enable the entire process by using the `enabled="true"` attribute. After this attribute is enabled, you work with the `<databases>` section. Nested within the `<databases>` nodes, you use the `<add>` element to reference the Northwind database. The following table explains all the attributes of the `<add>` element.

Attribute	Description
name	The name attribute provides an identifier to the SQL Server database.
connectionStringName	The connectionStringName attribute specifies the name of the connection. Because the connection string in the preceding example is called AppConnectionString1, you use this value for the connectionStringName attribute as well.
pollTime	The pollTime attribute specifies the time interval from one SQL Server poll to the next. The default is 5 seconds or 500 milliseconds (as shown in the previous example).

Now that the `web.config` file is set up correctly, you can start using SQL Server cache invalidation on your pages. ASP.NET makes a separate SQL Server request on a completely different thread to the `AspNet_SqlCacheTablesForChangeNotification` table to see if the `changeId` number has been incremented. If the number is changed, ASP.NET knows that an underlying change has been made to the SQL Server table and that a new result set should be retrieved. When it checks to see if it should make a SQL Server call, the request to the small `AspNet_SqlCacheTablesForChangeNotification` table has a single result. This is done in such a quick fashion that you can notice the difference in speed with SQL Server cache invalidation enabled.

Testing SQL Server Cache Invalidation

Now that the `web.config` file is set up and ready to go, the next step is to actually apply these new capabilities to a page. For an example of a page using the new SQL Server cache invalidation process, look at Listing 11-2.

Listing 11-2: An ASP.NET page utilizing SQL Server cache invalidation

VB

```
<%@ Page Language="VB" %>
<%@ OutputCache Duration="3600" VaryByParam="none"
    SqlDependency="Northwind:Customers"%>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Page created on " & DateTime.Now.ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label><br />
        <br />
        <asp:GridView ID="GridView1" Runat="server" DataSourceID="SqlDataSource1">
        </asp:GridView>
```

```

        <asp:SqlDataSource ID="SqlDataSource1" Runat="server"
            SelectCommand="Select * From Customers"
            ConnectionString="<%= ConnectionStrings:AppConnectionString1 %>"
            ProviderName="<%= ConnectionStrings:AppConnectionString1.providername %>"
        </asp:SqlDataSource>
    </form>
</body>
</html>
C#
<%@ Page Language="C#" %>
<%@ OutputCache Duration="3600" VaryByParam="none"
    SqlDependency="Northwind:Customers"%>

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        Label1.Text = "Page created on " + DateTime.Now.ToString();
    }
</script>

```

The first and most important part of this page is the `OutputCache` page directive that is specified at the top of the file. Typically, the `OutputCache` directive specifies how long the page output is held in the cache using the `Duration` attribute. It is followed by the `VaryByParam` attribute, which does not permit separate page outputs to be cached based on factors like the requestor's browser. The new addition is the `SqlDependency` attribute. This enables a particular page to use SQL Server cache invalidation. The following line shows the format of the value for the `SqlDependency` attribute:

```
SqlDependency="database:table"
```

The value of `Northwind:Customers` specifies that you want the SQL Server cache invalidation enabled for the `Customers` table within the `Northwind` database. The `Duration` attribute of the `OutputCache` directive shows you that, typically, the output of this page is stored in the cache for a long time — but this cache is disabled if the `Customers` table has any underlying changes made to the data that it contains.

A change to any of the cells in the `Customers` table of the `Northwind` database invalidates the cache, and a new cache is generated from the results, which now contain a new SQL Server database request. Figure 11-9 shows an example of the page generated the first time it is run.

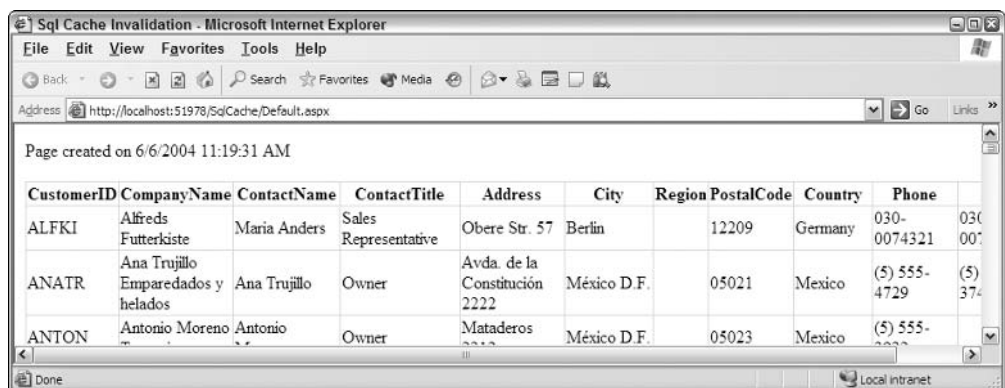
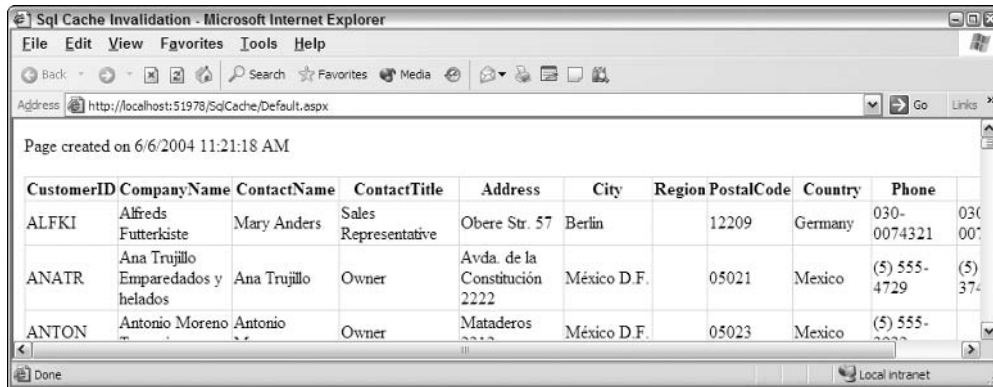


Figure 11-9

Chapter 11

From this figure, you can see the contents of the customer with the CustomerID of ALFKI. For this entry, you go to SQL Server and change the value of the ContactName from Maria Anders to Mary Anders. Before SQL Server cache invalidation, this change would have done nothing to the output cache. The original page output in the cache would still be present and the end user would still see the Maria Anders entry for the duration specified in the page's OutputCache directive. Because of SQL Server cache invalidation, after the underlying information in the table is changed, the output cache is invalidated, a new result set is retrieved, and the new result set is cached. When a change has been made, you see the results shown in Figure 11-10.



CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone
ALFKI	Alfreds Futterkiste	Mary Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729
ANTON	Antonio Moreno	Antonio	Owner	Mataderos	México D.F.		05023	Mexico	(5) 555-2000

Figure 11-10

Adding more than one table to a page

The preceding example shows how to use SQL Server cache invalidation for a single table on the ASP.NET page. What do you do if your page is working with two or more tables?

To add more than one table, you use the OutputCache directive shown here:

```
SqlDependency="database:table;database:table"
```

From this example, you can see that the value of the SqlDependency attribute separates the databases and tables with a semicolon. If you want to work with both the Customers table and the Products table of the Northwind database, you construct the value of the SqlDependency attribute as follows:

```
SqlDependency="Northwind:Customers;Northwind:Products"
```

Attaching SQL Server cache dependencies to the Request object

In addition to changing settings in the OutputCache directive to activate SQL Server cache invalidation, you can also set the SQL Server cache invalidation programmatically. To do so, you use the SqlCacheDependency class, which is illustrated in Listing 11-3.

Listing 11-3: Working with SQL Server cache invalidation programmatically**VB**

```
Dim myDependency As SqlCacheDependency = _
    New SqlCacheDependency("Northwind", "Customers")
Response.AddCacheDependency(dependency)
Response.Cache.SetValidUntilExpires(true)
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60))
Response.Cache.SetCacheability(HttpCacheability.Public)
```

C#

```
SqlCacheDependency myDependency = new SqlCacheDependency("Northwind", "Customers");
Response.AddCacheDependency(myDependency);
Response.Cache.SetValidUntilExpires(true);
Response.Cache.SetExpires(DateTime.Now.AddMinutes(60));
Response.Cache.SetCacheability(HttpCacheability.Public);
```

You first create an instance of the `SqlCacheDependency` object — assigning it the value of the database and the table at the same time. The `SqlCacheDependency` class takes the following parameters:

```
SqlCacheDependency(databaseEntryName As String, tablename As String)
```

You use this parameter construction if you are working with SQL Server 7.0 or with SQL Server 2000. If you are working with SQL Server “Yukon,” you use the following construction:

```
SqlCacheDependency(sqlCmd As System.Data.SqlClient.SqlCommand)
```

After the `SqlCacheDependency` class is in place, you add the dependency to the `Cache` object and set some of the properties of the `Cache` object as well. You can do this either programmatically or through the `OutputCache` directive.

Attaching SQL Server cache dependencies to the Cache object

In addition to attaching SQL Server cache dependencies to the `Request` object, you can attach them to the `Cache` object as well. The `Cache` object is contained within the `System.Web.Caching` namespace, and it enables you to work programmatically with the caching of any type of objects. Listing 11-4 shows a page that utilizes the `Cache` object with the `SqlDependency` object.

Listing 11-4: Using the Cache object with the SqlDependency object**VB**

```
<%@ Page Language="VB" %>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.SqlClient"%>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        Dim myCustomers As DataSet
        myCustomers = CType(Cache("firmCustomers", DataSet))
```

(continued)

Listing 11-4: *(continued)*

```
If myCustomers Is Nothing Then
    Dim conn As SqlConnection = _
        New SqlConnection(ConfigurationSettings.ConnectionStrings("Northwind"))
    Dim da As SqlDataAdapter = _
        New SqlDataAdapter("Select * From Customers", conn)

    myCustomers = New DataSet
    da.Fill(myCustomers)

    Dim myDependency As SqlCacheDependency = _
        New SqlCacheDependency("Northwind", "Customers")
    Cache.Insert("firmCustomers", myCustomers, myDependency)

    Label1.Text = "Produced from database."
Else
    Label1.Text = "Produced from Cache object."
End If

GridView1.DataSource = myCustomers
GridView1.DataBind()
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Sql Cache Invalidation</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Label ID="Label1" Runat="server"></asp:Label><br />
        <asp:GridView ID="GridView1" Runat="server">
        </asp:GridView>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data"%>
<%@ Import Namespace="System.Data.SqlClient"%>

<script runat="server">
    void Page_Load(object sender, System.EventArgs e)
    {
        DataSet myCustomers;
        myCustomers = (DataSet)Cache["firmCustomers"];

        if (categories == null)
        {
            SqlConnection conn = new
                SqlConnection(ConfigurationSettings.ConnectionStrings["Northwind"]);
            SqlDataAdapter da = new
```

```

        SqlDataAdapter("Select * from Customers", conn);

        myCustomers = new DataSet();
        da.Fill(myCustomers);

        SqlCacheDependency myDependency = new
            SqlCacheDependency("Northwind", "Customers");
        Cache.Insert("firmCustomers", myCustomers, myDependency);

        Label1.Text = "Produced from database.";
    }
    else
    {
        Label1.Text = "Produced from Cache object.";
    }

    GridView1.DataSource = myCustomers;
    GridView1.DataBind();
}
</script>

```

In this example, the `SqlCacheDependency` class associated itself to the `Customers` table in the `Northwind` database as before. This time, however, you use the `Cache` object to insert the retrieved dataset along with a reference to the `SqlCacheDependency` object. The `Insert` method of the `Cache` class is constructed as follows:

```

Cache.Insert(key As String, value As Object,
    dependencies As System.Web.Caching.CacheDependency)

```

You can also insert more information about the dependency using the following construct:

```

Cache.Insert(key As String, value As Object,
    dependencies As System.Web.Caching.CacheDependency
    absoluteExpiration As Date, slidingExpiration As System.TimeSpan)

```

And finally:

```

Cache.Insert(key As String, value As Object,
    dependencies As System.Web.Caching.CacheDependency
    absoluteExpiration As Date, slidingExpiration As System.TimeSpan)
    priority As System.Web.Caching.CacheItemPriority,
    onRemoveCallback As System.Web.Caching.CacheItemRemovedCallback)

```

The SQL Server cache dependency created comes into action and does the same polling as it would have done otherwise. If any of the data in the `Customers` table has changed, the `SqlCacheDependency` class invalidates what is stored in the cache. On the next request, the `Cache("firmCustomers")` is found to be empty and a new request is made to SQL Server. The `Cache` object again repopulates the cache with the new results generated.

When the ASP.NET page from Listing 11-4 is called for the first time, the results generated are as shown in Figure 11-11.

Produced from database.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076543
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See	Hanna Moos	Sales	Friedrichstr. 57	Münchheim		68306	Germany	0621-0621	0621-0621

Figure 11-11

Because this is the first time that the page is generated, nothing is in the cache. The Cache object is, therefore, placed in the result set along with the association to the SQL Server cache dependency. Figure 11-12 shows the result for the second request.

Produced from Cache object.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	030-0074321	030-0076543
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taqueria	Antonio Moreno	Owner	Mataderos 2312	México D.F.		05023	Mexico	(5) 555-3932	
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London		WA1 1DP	UK	(171) 555-7788	(171) 555-6750
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		S-958 22	Sweden	0921-12 34 65	0921-12 34 67
BLAUS	Blauer See	Hanna Moos	Sales	Friedrichstr. 57	Münchheim		68306	Germany	0621-0621	0621-0621

Figure 11-12

On the second request, the dataset is already contained within the cache and, therefore, it is retrievable. You aren't required to hit SQL Server to get the full results again. If any of the information has changed within SQL Server itself, however, the Cache object returns nothing, and a new result set is retrieved.

Summary

SQL Server cache invalidation is an outstanding new feature of ASP.NET 2.0 that enables you to invalidate items stored in the cache when underlying changes occur to the data in the tables being monitored.

When you are monitoring changes to the database, you can configure these procedures easily in the `web.config` file, or you can work programmatically with cache invalidation directly in your code. These changes are possible because the `CacheDependency` object has been unsealed. You can now inherit from this object and create your own cache dependencies. The SQL Server cache invalidation process is the first example of this capability.

12

Additional New Controls

When I sat in one of the first review sessions for ASP.NET 2.0 on the Microsoft campus in Redmond, Washington, I remember being amazed by the number of new server controls (in addition to many other new and exciting features) this newest release offered. The core infrastructure was already in place with ASP.NET 1.0/1.1; but with the much improved 2.0 release, the ASP.NET team was making the lives of developers even simpler.

The purpose of this large collection of new controls is to make you more productive. They enable you to introduce advanced functionality that would have been laboriously programmed or simply omitted in the past. For example, in the classic ASP days, few calendars were used on Internet Web sites. With the introduction of the Calendar server control in ASP.NET 1.0, calendar creation on a site became a trivial task. And with ASP.NET 1.0/1.1, it was rather difficult to build an image map on top of an image. Through the use of a new server control, this is now built into ASP.NET 2.0.

I covered a considerable number of these new controls in the preceding chapters, but I still have quite a few new server controls to discuss. This chapter takes a look at some of these new server controls and explains how to use them in ASP.NET 2.0 applications.

BulletedList Server Control

One common HTML Web page element is a collection of items in a bulleted list. The BulletedList server control is meant to easily display a bulleted list of items in an ordered (using the HTML `` element) or unordered (using the HTML `` element) fashion.

In addition to creating lists that are ordered or unordered, you can use this control to determine the style used for displaying the list.

Chapter 12

The BulletedList control can be constructed of any number of `<asp:ListItem>` controls or be data-bound to a data source of some kind and populated based upon the contents retrieved. Listing 12-1 shows a bulleted list in its simplest form.

Listing 12-1: A simple BulletedList control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>BulletedList Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:BulletedList ID="Bulletedlist1" Runat="server">
      <asp:ListItem>United States</asp:ListItem>
      <asp:ListItem>United Kingdom</asp:ListItem>
      <asp:ListItem>Finland</asp:ListItem>
      <asp:ListItem>Russia</asp:ListItem>
      <asp:ListItem>Sweden</asp:ListItem>
      <asp:ListItem>Estonia</asp:ListItem>
    </asp:BulletedList>
  </form>
</body>
</html>
```

The use of the `<asp:BulletedList>` element, along with the `<asp:ListItem>` elements, produces a simple bulleted list output like the one shown in Figure 12-1.

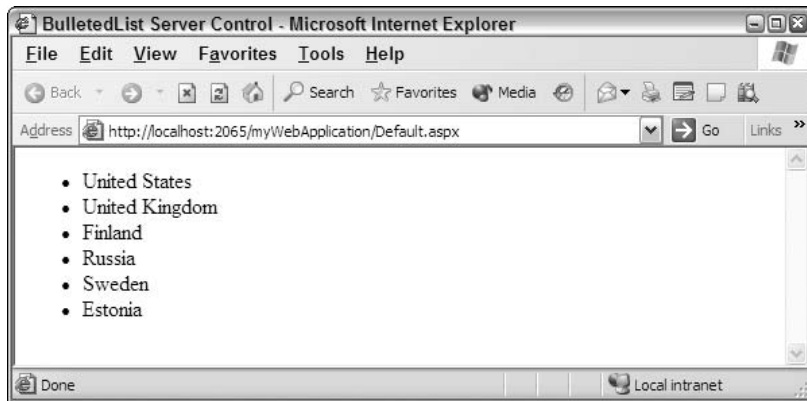


Figure 12-1

The BulletedList control also enables you to easily change the style of the list with just one or two attributes. The `BulletStyle` attribute changes the style of the bullet that precedes each line of the bulleted list. This attribute has possible values of `Numbered`, `LowerAlpha`, `UpperAlpha`, `LowerRoman`, `UpperRoman`, `Disc`, `Circle`, `Square`, `NotSet`, and `CustomImage`. Figure 12-2 shows an example of these styles (minus the `CustomImage` setting, as it can be any image of your choice).

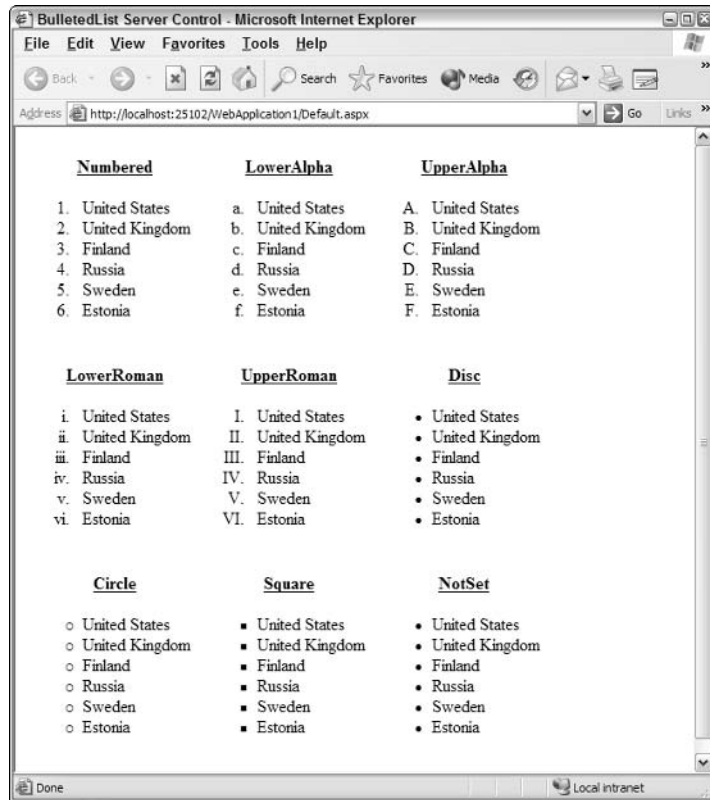


Figure 12-2

When working with any of the numbered styles (Numbered, LowerAlpha, UpperAlpha, LowerRoman, UpperRoman), you can also change the starting value of the first bulleted item in the list by using the `FirstBulletNumber` attribute. Setting this attribute's value to 5 when you use the `UpperRoman` setting results in the format illustrated in Figure 12-3.

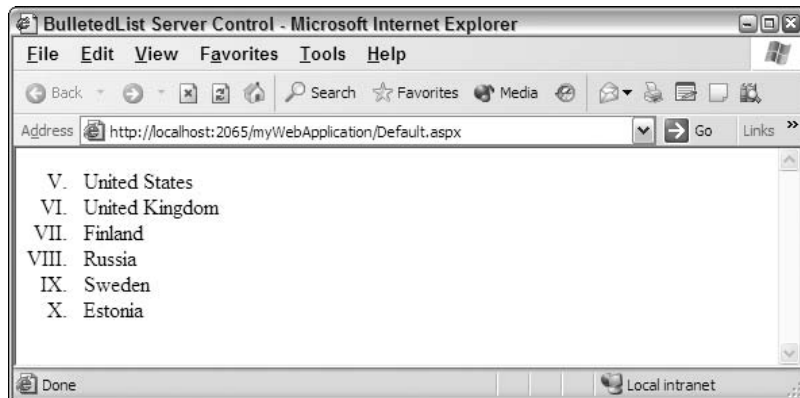


Figure 12-3

Chapter 12

When you use the `CustomImage` setting in the `BulletedList` control, you must also use the `BulletedImageUrl` attribute in the following manner:

```
<asp:BulletedList ID="Bulletedlist1" Runat="server"
    BulletStyle="CustomImage" BulletedImageUrl="~/myImage.gif">
```

The `BulletedList` control has an attribute called `DisplayMode`, which has three possible values: `Text`, `HyperLink`, and `LinkButton`. `Text` is the default and has been used so far in the examples. Using `Text` means that the items in the bulleted list are laid out only as text. `HyperLink` means that each of the items will be turned into a hyperlink — any user clicking the link is redirected to another page. This page is specified by the `<asp:ListItem>` control's `Value` attribute. A value of `LinkButton` turns each bulleted list item into a hyperlink that posts back to the same page. It enables you to retrieve the selection that the end user makes, as illustrated in Listing 12-2.

Listing 12-2: Using the `LinkButton` value for the `DisplayMode` attribute

VB

```
<%@ Page Language="VB"%>
```

```
<script runat="server">
    Sub BulletedList1_Click(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.BulletedListEventArgs)

        Label1.Text = "The index of item you selected: " & e.Index & _
            "The value of the item selected: " & _
            BulletedList1.Items(e.Index).Text
    End Sub
</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>BulletedList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:BulletedList ID="BulletedList1" Runat="server"
            OnClick="BulletedList1_Click" DisplayMode="LinkButton">
            <asp:ListItem>United States</asp:ListItem>
            <asp:ListItem>United Kingdom</asp:ListItem>
            <asp:ListItem>Finland</asp:ListItem>
            <asp:ListItem>Russia</asp:ListItem>
            <asp:ListItem>Sweden</asp:ListItem>
            <asp:ListItem>Estonia</asp:ListItem>
        </asp:BulletedList>
        <asp:Label ID="Label1" Runat="server">
        </asp:Label>
    </form>
</body>
</html>
```

C#

```
<script runat="server">
    void BulletedList1_Click(object sender,
        System.Web.UI.WebControls.BulletedListEventArgs e)
    {
        Label1.Text = "The index of item you selected: " + e.Index +
            "The value of the item selected: " +
            BulletedList1.Items[e.Index].Text;
    }
</script>
```

In this example, the `DisplayMode` attribute is set to `LinkButton`, and the `OnClick` attribute is used to point to the `BulletedList1_Click` event. This event uses the `BulletedListEventArgs` object, which only exposes the `Index` property. Using this, you can determine the index number of the item selected.

You can directly access the `Text` value of a selected item by using the `Items` property, or you can use the same method to populate an instance of the `ListItem` object. You do so as shown here:

VB

```
Dim blSelectedValue As ListItem = BulletedList1.Items(e.Index)
```

C#

```
ListItem blSelectedValue = BulletedList1.Items[e.Index];
```

Now that you have seen how to create bulleted lists with items that you declaratively place in the code, take a look at how to create dynamic bulleted lists from items that are stored in a data store of some kind. For an example of how to use the `BulletedList` control to data-bind to results coming from a data store, look at the following example where all information is retrieved from an XML file.

The first step is to create the XML file. For this example, I use the XML file shown in Listing 12-3.

Listing 12-3: FilmChoices.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FilmChoices>
  <Film>
    <Title>Close Encounters of the Third Kind</Title>
    <Year>1977</Year>
    <Director>Steven Spielberg</Director>
  </Film>
  <Film>
    <Title>Grease</Title>
    <Year>1978</Year>
    <Director>Randal Kleiser</Director>
  </Film>
  <Film>
    <Title>Lawrence of Arabia</Title>
    <Year>1962</Year>
    <Director>David Lean</Director>
  </Film>
</FilmChoices>
```

To populate the BulletedList server control with the <Title> element from the FileChoices.xml file, use a DataSetDataSource control to access the file, as illustrated in Listing 12-4.

Listing 12-4: Dynamically populating a BulletedList server control

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>BulletedList Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:BulletedList ID="Bulletedlist1" Runat="server"
      DataSourceId="DataSetDataSource1" DataTextField="Title">
    </asp:BulletedList>
    <asp:DataSetDataSource ID="DataSetDataSource1"
      Runat="server" DataFile="~/FilmChoices.xml">
    </asp:DataSetDataSource>
  </form>
</body>
</html>
```

In this example, you use the DataSourceId attribute to point to the DataSetDataSource control (as you would with any control that can be bound to one of the data source controls). After you are connected to this data source control, you specifically point at the <Title> element by using the DataTextField attribute. After the two server controls are connected and the page is run, you get a bulleted list that is completely generated from the contents of the XML file. The result is shown in Figure 12-4.



Figure 12-4

HiddenField Server Control

For many years now, developers have been using hidden fields in their Web pages to work with state management. The <input type="hidden"> element is ideal for storing items that have no security context to them. These items are simply placeholders for data points that you want to store in the page itself instead of using the Session object or intermingling the data with the view state of the page. View

state is another great way to store information in a page, but many developers turn off this feature in order to avoid corruption of the view state or possibly degradation of page performance.

Any time a hidden field is placed within a Web page, it is not interpreted in the browser in any fashion, although is completely viewable by end users if they look at the source of the HTML page.

Listing 12-5 is an example of using the HiddenField server control to hold a GUID that can be used from page to page simply by carrying over its value as the end user navigates through your application.

Listing 12-5: Working with the HiddenField server control

VB

```
<%@ Page Language="VB" %>

<script runat="server" language="vb">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        HiddenField1.Value = System.Guid.NewGuid().ToString()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>HiddenField Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:HiddenField ID="HiddenField1" Runat="Server" />
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
        HiddenField1.Value = System.Guid.NewGuid().ToString();
    }
</script>
```

In this example, the Page_Load event populates the HiddenField1 control with a GUID. You can see the hidden field and the value created by looking at the source of the blank HTML page that is created. You should see a result similar to the following (the GUID has a different value, of course):

```
<input type="hidden" name="HiddenField1" id="HiddenField1"
value="a031e77c-379b-4b4a-887c-244ee69584d5" />
```

On the page postback, ASP.NET can detect whether the HiddenField server control has changed its value since the last post. This enables you to change the HiddenField value with client-side script and then work with the changes in a page event.

Chapter 12

The HiddenField server control has an event called `ValueChanged` that you can use when the value is changed:

VB

```
Sub HiddenField1_ValueChanged(ByVal sender As Object, ByVal e As System.EventArgs)
    ' Handle event here
End Sub
```

C#

```
void HiddenField1_ValueChanged(object sender, EventArgs e)
{
    // Handle event here
}
```

This `ValueChanged` event is triggered when the ASP.NET page gets posted back to the server if the value of the HiddenField server control has changed since the last time the page was drawn. If the value of the HiddenField control has not changed, this method is never triggered. Therefore, this method is useful when you act upon any changes to the HiddenField control — such as recording a value to the database or changing a value in the user's profile.

FileUpload Server Control

In ASP.NET 1.0/1.1, it was quite possible to upload files using the HTML FileUpload control. This HTML server control put an `<input type="file">` element on your Web page to enable the end user to upload files to the server. In order to use this file, however, you had to make a couple of modifications to the page. You were, for example, required to add an `enctype="multipart/form-data"` to the page's `<form>` element.

ASP.NET 2.0 introduces a new FileUpload server control that makes the process of uploading files to a server even simpler than before. When giving a page the capability to upload files, you simply include the new `<asp:FileUpload>` control and ASP.NET takes care of the rest, including adding the `enctype` attribute to the page's `<form>` element for you.

After the file is uploaded to the server, you can also take hold of the uploaded file's properties and either display them to the end user or use these values yourself in your page's code-behind. Listing 12-6 shows an example of using the new FileUpload control. The page contains a single FileUpload control, plus a Button and a Label control.

Listing 12-6: Uploading files using the new FileUpload control

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If FileUpload1.HasFile Then
            Try
                FileUpload1.SaveAs("C:\Uploads\" & _
                    FileUpload1.FileName)
                Label1.Text = "File name: " & _
```

```

        FileUpload1.PostedFile.FileName & "<br>" & _
        "File Size: " & _
        FileUpload1.PostedFile.ContentLength & " kb<br>" & _
        "Content type: " & _
        FileUpload1.PostedFile.ContentType
    Catch ex As Exception
        Label1.Text = "ERROR: " & ex.Message.ToString()
    End Try
Else
    Label1.Text = "You have not specified a file."
End If
End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>FileUpload Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:FileUpload ID="FileUpload1" Runat="server" />
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Upload"
                OnClick="Button1_Click" /></p>
        <p>
            <asp:Label ID="Label1" Runat="server"></asp:Label></p>
    </form>
</body>
</html>

```

C#

```

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

<script runat="server">
    void Button1_Click(object sender, EventArgs e)
    {
        if (FileUpload1.HasFile)
        {
            try {
                FileUpload1.SaveAs("C:\\Uploads\\" + FileUpload1.FileName);
                Label1.Text = "File name: " +
                    FileUpload1.PostedFile.FileName + "<br>" +
                    FileUpload1.PostedFile.ContentLength + " kb<br>" +
                    "Content type: " +
                    FileUpload1.PostedFile.ContentType;
            }
            catch (Exception ex) {
                Label1.Text = "ERROR: " + ex.Message.ToString();
            }
        }
        else
        {
            Label1.Text = "You have not specified a file.";
        }
    }
</script>

```

Chapter 12

From this example, you can see that the entire process is rather simple. The single button on the page initiates the upload process. The first check done examines whether a file reference was actually placed within the `<input type="file">` element. If a file was specified, an attempt is made to upload the referenced file to the server using the `SaveAs` method of the `FileUpload` control. This method takes a single `String` parameter, which should include the location where you want to save the file. In the `String` parameter that I use in Listing 12-6, you can see that I am saving the file to a folder called `Uploads`, which is located in the `C:\` drive.

In addition to saving the file uploaded to the `C:\Uploads` folder, I give the saved file the same name as the file it was copied from. To do this, I use the `PostedFile.FileName` attribute. If you want to name the file something else, you simply use the `SaveAs` method in the following manner:

```
FileUpload1.SaveAs("C:\Uploads\UploadedFile.txt")
```

You could also give the file a name that specifies the time it was uploaded:

```
FileUpload1.SaveAs("C:\Uploads\" & System.DateTime.Now.ToFileTimeUtc() & ".txt")
```

In this case, the file is named according to the time the upload occurred.

After the upload is successfully completed, the `Label` control on the page is populated with metadata of the uploaded file. In this case, the file's name, size, and content type are retrieved and displayed on the page for the end user. When the file is uploaded to the server, the page generated is similar to that shown in Figure 12-5.



Figure 12-5

Uploading files to another server can be an error-prone affair. It is vital to upload files in your code using proper exception handling. This is why the file in the example is uploaded using a `Try Catch` statement. Another way to prevent the `FileUpload` control from ruining the experience of the end user is to use the `FileUpload`'s `AlternateText` attribute:

```
<asp:FileUpload ID="FileUpload1" Runat="server"
  AlternateText="Your browser does not allow uploads." />
```

This attribute assists end users whose browsers are unable to understand the `<input type="file">` element. That specific user gets the text shown in the `AlternateText` attribute.

MultiView and View Server Controls

The MultiView and View server controls work together to give you the capability to turn on and off sections of an ASP.NET page. Turning sections on and off, which means activating or deactivating a series of View controls within a MultiView control, is similar to changing the visibility of Panel controls. For certain operations, however, you may find that the MultiView control is easier to manage and work with.

The sections, or views, do not change on the client-side; rather, they change with a postback to the server. In each view, you can put any number of elements and controls, and the end user can work through the views based upon the sequence numbers that you assign to the views.

You can build them (like all server controls) from the Source view or Design view. If working with Visual Studio 2005, you can drag and drop a MultiView control onto the design surface and then drag and drop any number of View controls inside the MultiView control. You place the elements you want within the View controls. After this is completed, you have something like the view in Figure 12-6.

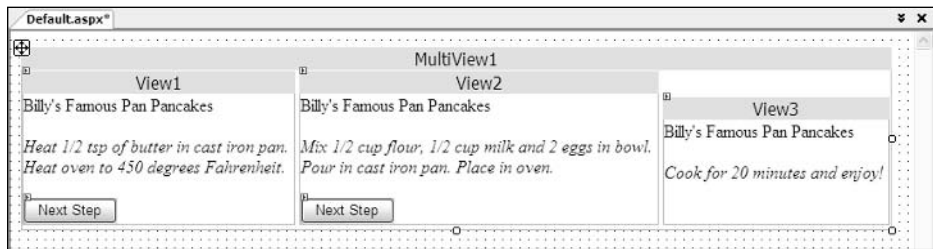


Figure 12-6

The other option is to create this directly in the code, as shown in Listing 12-7.

Listing 12-7: Using the MultiView and View server controls

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        If Not Page.IsPostBack Then
            MultiView1.ActiveViewIndex = 0
        End If
    End Sub

    Sub NextView(ByVal sender As Object, ByVal e As System.EventArgs)
        MultiView1.ActiveViewIndex += 1
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```

(continued)

Listing 12-7: (continued)

```
<title>MultiView Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:MultiView ID="MultiView1" Runat="server">
      <asp:View ID="View1" Runat="Server">
        Billy's Famous Pan Pancakes<p />
        <i>Heat 1/2 tsp of butter in cast iron pan.<br />
        Heat oven to 450 degrees Fahrenheit.<br />
        </i><p />
        <asp:Button ID="Button1" Runat="Server" Text="Next Step"
          OnClick="NextView" />
      </asp:View>
      <asp:View ID="View2" Runat="Server">
        Billy's Famous Pan Pancakes<p />
        <i>Mix 1/2 cup flour, 1/2 cup milk and 2 eggs in bowl.<br />
        Pour in cast iron pan. Place in oven.</i><p />
        <asp:Button ID="Button2" Runat="Server" Text="Next Step"
          OnClick="NextView" />
      </asp:View>
      <asp:View ID="View3" Runat="Server">
        Billy's Famous Pan Pancakes<p />
        <i>Cook for 20 minutes and enjoy!<br />
        </i><p />
      </asp:View>
    </asp:MultiView>
  </form>
</body>
</html>
```

C#

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

<script runat="server">
  void Page_Load(object sender, EventArgs e)
  {
    if (!Page.IsPostBack)
    {
      MultiView1.ActiveViewIndex = 0;
    }
  }

  void NextView(object sender, EventArgs e)
  {
    MultiView1.ActiveViewIndex += 1;
  }
</script>
```

In the example in Listing 12-7, you can see that three views are expressed in the MultiView control. Each view is constructed with an `<asp:View>` server control that also needs `ID` and `Runat` attributes added. A button is added to each of the first two views (`View1` and `View2`) of the MultiView control. These buttons both point to a server-side event that triggers the MultiView control to progress onto the next view within the series of views.

Before either of the buttons can be clicked, the MultiView control's `ActiveViewIndex` attribute is assigned a value. By default, the `ActiveViewIndex`, which describes the view that should be showing, is set to `-1`. This means that no view shows when the page is generated. To start on the first view when the page is drawn, you set the `ActiveViewIndex` property to `0`, which is the first view because this is a zero-based index. Therefore, the code from Listing 12-7 first checks to see if the page is in a postback situation and if not, the `ActiveViewIndex` is assigned to the first View control.

Each of the buttons in the MultiView control triggers the `NextView` method. This method simply adds one to the `ActiveViewIndex` value — thereby showing the next view in the series until the last view is shown. The view series is illustrated in Figure 12-7.

In addition to the Next Step button on the first two views, you can also place a button in the last two views that allows the user to navigate backward through the views. To do this, create two buttons titled Previous Step in the last two views, both of which point to the following method in their `OnClick` events:

VB

```
Sub PreviousView(ByVal sender As Object, ByVal e As System.EventArgs)
    MultiView1.ActiveViewIndex -= 1
End Sub
```

C#

```
void PreviousView(object sender, EventArgs e)
{
    MultiView1.ActiveViewIndex -= 1;
}
```

Here, the `PreviousView` method subtracts one from the `ActiveViewIndex` value, thereby showing the previous view in the view series.

Another option to spice up the MultiView control is to add a step counter that displays to a Label control which step the end user is currently performing in the series. In the `Page_PreRender` event, you add the following line:

VB

```
Label1.Text = "Step " & (MultiView1.ActiveViewIndex + 1).ToString() & _
    " of " & MultiView1.Views.Count.ToString()
```

C#

```
Label1.Text = "Step " + (MultiView1.ActiveViewIndex + 1).ToString() +
    " of " + MultiView1.Views.Count.ToString();
```

Now when working through the MultiView control, the end user sees `Step 1 of 3` on the first view, which changes to `Step 2 of 3` on the next view, and so on.

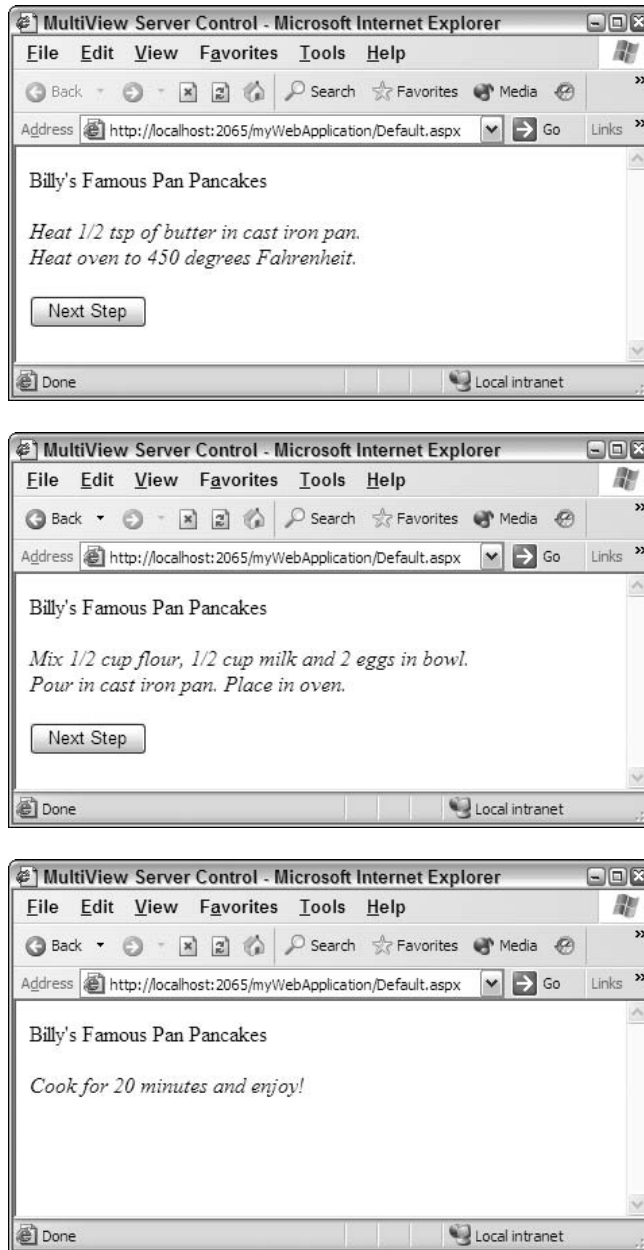


Figure 12-7

Wizard Server Control

Quite similar to the MultiView control, the Wizard server control enables you to build a sequence of steps that are displayed to the end user. Web pages are all about either displaying or gathering information and, in many cases, you don't want to display all the information at once — nor do you always want to gather everything from the end user at once. Sometimes, you want to trickle the information in from or out to the end user.

When you are constructing a step-by-step process that includes logic on the steps taken, use the Wizard control to manage the entire process. When working with the Wizard control for the first time, notice that this control allows for a far greater degree of customization than does the MultiView control.

In its simplest form, the Wizard control can be just an `<asp:Wizard>` element with any number of `<asp:WizardStep>` elements. In Listing 12-8, you create a Wizard control that works through three steps.

Listing 12-8: A simple Wizard control

```
<%@ Page Language="VB"%>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Wizard server control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Wizard ID="Wizard1" Runat="server" SideBarEnabled="true"
            ActiveStepIndex="0">
            <WizardSteps>
                <asp:WizardStep Runat="server" Title="Step 1">
                    This is the first step.</asp:WizardStep>
                <asp:WizardStep Runat="server" Title="Step 2">
                    This is the second step.</asp:WizardStep>
                <asp:WizardStep Runat="server" Title="Step 3">
                    This is the third and final step.</asp:WizardStep>
            </WizardSteps>
        </asp:Wizard>
    </form>
</body>
</html>
```

In this example, three steps are defined with the `<asp:WizardSteps>` control. Each step contains content. In this case, the content is simply text, but you can put anything you desire — such as other Web server controls or even user controls. The order in which the Wizard Steps are defined is completely based upon the order in which they appear within the `<WizardSteps>` element. Changing this order changes the order in which the end user sees them.

The `<asp:Wizard>` element itself contains a couple of important attributes. The first is the `SideBarEnabled` attribute. In this example, it is set to `True` — meaning that a side navigation system in the displayed control enables the end user to quickly navigate to other steps in the process. The `ActiveStepIndex` attribute of the Wizard control defines the first Wizard Step. In this case, it is the first step — 0.

Chapter 12

The three steps of this Wizard control are shown in Figure 12-8.

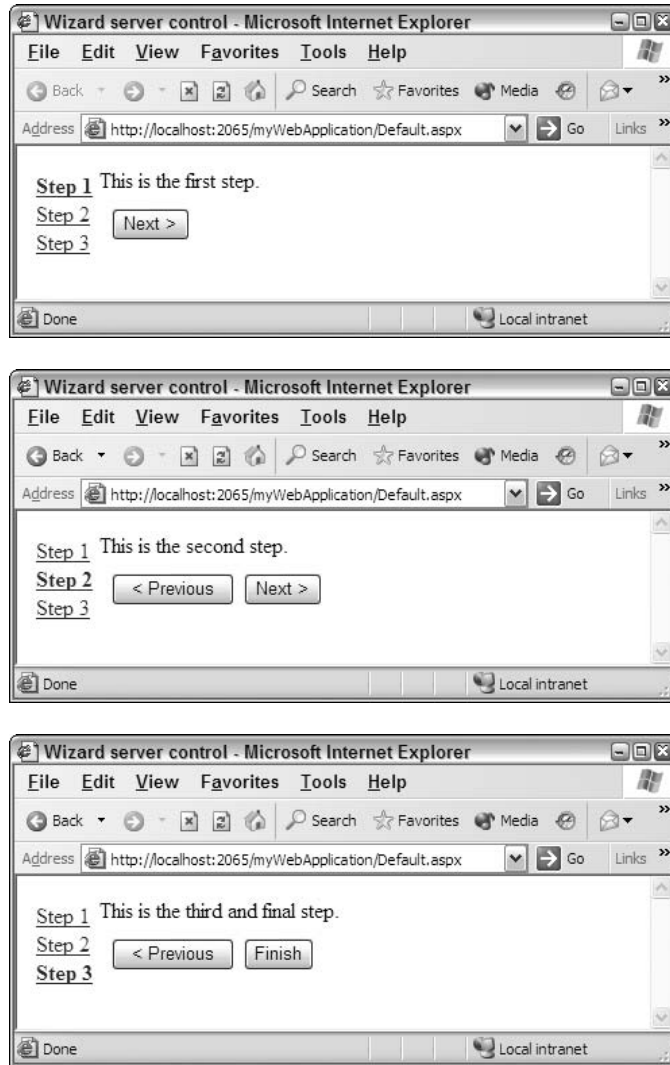


Figure 12-8

From this example, you can see that the side navigation allows for easy access to the steps defined. The Wizard control also adds appropriate buttons to the steps in the process. The first step has simply a Next button, whereas the middle step has Previous and Next buttons, and the final step has a Previous and a Finish button. Therefore, the end user can navigate through the steps using either the side navigation or the buttons on each of the steps. You can customize the Wizard control in so many ways that it tends to remind me of the other rich Web server controls from ASP.NET, such as the Calendar control. Because so much is possible, I cover only a few of the basics that you are most likely to employ in some of the Wizard controls you build.

Customizing the side navigation

From the example in Figure 12-8, you can see that the steps are defined as Step 1, Step 2, and Step 3. The links are created based upon the `Title` property's value that you give to each of the `<asp:WizardStep>` elements in the Wizard control:

```
<asp:WizardStep Runat="server" Title="Step 1">
  This is the first step.</asp:WizardStep>
```

By default, when creating Wizard Steps in Design view, each Wizard Step created is titled `Step x` (with `x` being the number in the sequence). You can easily change the value of the `Title` attributes of each of the Wizard Steps to define the steps as you see fit. Figure 12-9 shows the side navigation of the Wizard control with renamed titles.

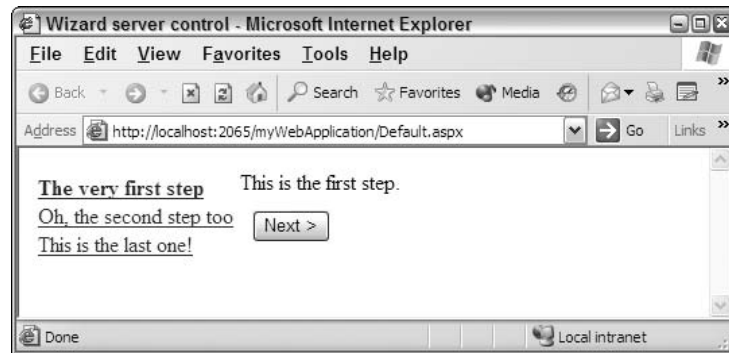


Figure 12-9

Examining the AllowReturn attribute

Some other interesting points of customization for the side navigation piece of the Wizard control include the `AllowReturn` attribute. By setting this attribute on one of the Wizard Steps to `False`, you can remove the capability for the end users to go back to the step after they have viewed it. This ensures that the end user cannot backward navigate to any of the viewed steps that contained this attribute, but he would be able to return to any steps that do not contain this attribute or which have this attribute set to `True`:

```
<asp:WizardStep Runat="server" Title="Step 1" AllowReturn="False">
  This is the first step.</asp:WizardStep>
```

Working with the StepType attribute

Another interesting attribute in the `<asp:WizardStep>` element is the `StepType` attribute. The `StepType` attribute defines the structure of the buttons used on the steps. For instance, by default, the Wizard control places only a Next button on the first step. It understands that because this is the first step, you probably don't need the Previous button here. It also knows to use a Next and Previous button on the middle step, but it uses a Previous and a Finish button on the last step. It draws the buttons in this fashion because, by default, the `StepType` attribute is set to `Auto`, meaning that the Wizard control determines the placement

of buttons. You can, however, take control of the `StepType` attribute in the `<asp:WizardStep>` element to make your own determination about which buttons are used for which steps.

Besides a `StepType` value of `Auto`, the other options include `Start`, `Step`, `Finish`, and `Complete`. A value of `Start` for the Wizard Step means that the step defined has only a Next button and nothing else. This simply allows the user to proceed to the next step in the series. A value of `Step` means that the Wizard Step has a Next and a Previous button. A value of `Finish` means that the step includes a Previous and a Finish button. The last one, `Complete`, is an interesting step that you can add to give some final message to the end user who is working through the steps of your Wizard control. In the Wizard control shown in Listing 12-8, for example, when the end user gets to the last step and clicks the Finish button, nothing happens and the user just stays on the last page. You can address this by adding a final step to the collection of Wizard Steps to give a final message as shown in Listing 12-9.

Listing 12-9: Having a Complete step in the Wizard Step collection

```
<WizardSteps>
  <asp:WizardStep Runat="server" Title="Step 1">
    This is the first step.</asp:WizardStep>
  <asp:WizardStep Runat="server" Title="Step 2">
    This is the second step.</asp:WizardStep>
  <asp:WizardStep Runat="server" Title="Step 3">
    This is the third and final step.</asp:WizardStep>
  <asp:WizardStep Runat="server" Title="Final Step" StepType="Complete">
    Thanks for working through the steps.</asp:WizardStep>
</WizardSteps>
```

When you run this Wizard control in a page, you still only see the first three steps in the side navigation. Because the last step has a `StepType` set to `Complete`, it does not appear in the side navigation list, and when the end user clicks the Finish button in Step 3, the last step — *Final Step* — is shown and no buttons are shown with it.

Adding a header to the Wizard control

The Wizard control enables you to place a header at the top of the control with the use of the `HeaderText` attribute in the main `<asp:Wizard>` element. This is illustrated in Listing 12-10.

Listing 12-10: Working with the HeaderText attribute

```
<asp:Wizard ID="Wizard1" Runat="server" SideBarEnabled="true" ActiveStepIndex="0"
  HeaderText="&nbsp;Step by Step with the Wizard control&nbsp;"
  HeaderStyle-BackColor="DarkGray" HeaderStyle-Font-Bold="true"
  HeaderStyle-Font-Size="20">

  ...

</asp:Wizard>
```

This code creates a header that appears on each of the steps in the Wizard. The result of this snippet is shown in Figure 12-10.

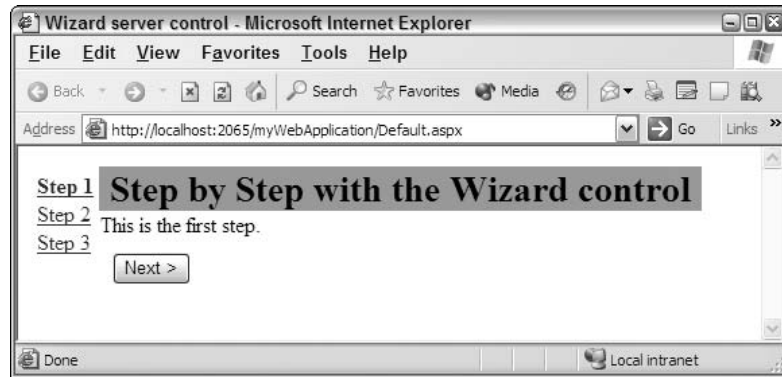


Figure 12-10

Working with the Wizard's navigation system

As I stated earlier, the Wizard control allows for a very high degree of customization — especially in the area of style. You can customize every single aspect of the process, as well as how every element appears to the end user.

Pay particular attention to the options that are available for customization of the navigation buttons. By default, the Wizard Steps use buttons for Next, Previous, and Finish that are used throughout the entire series of steps. From the main `<asp:Wizard>` element, you can change everything about these buttons and how they work.

First, if you look through the long list of attributes available for this element, notice that one available button isn't shown by default. The Cancel button can be added by setting the value of the `DisplayCancelButton` attribute to `True`. As shown in Figure 12-11, this gives you a Cancel button within the navigation created for each and every step (including the final step in the series).

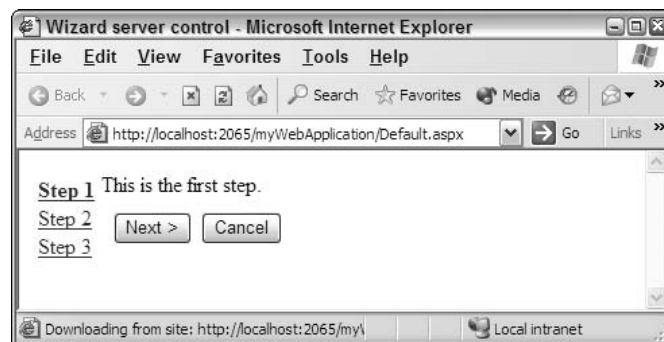


Figure 12-11

After you decide which buttons to use within the Wizard navigation, you can choose the style of these buttons. By default, regular buttons appear, but you can change the button style with the `CancelButtonType`, `FinishStepButtonType`, `FinishStepPreviousButtonType`, `NextStepButtonType`, `PreviousStepButtonType`, and `StartStepNextButtonType` attributes. All these attributes change the style of the buttons used for the navigation system of the Wizard control. If you use any of these button types and want all the buttons consistently styled, you must change each attribute to the same value. The possible values of these button-specific elements include `Button`, `Image`, and `Link`. `Button` is the default and means that the navigation system uses buttons. A value of `Image` enables you to use image buttons, and `Link` turns a selected item in the navigation system into a hyperlink.

In addition to these button-specific attributes of the `<asp:Wizard>` element, you can also specify a URL to which the user is directed when the he clicks either the Cancel or Finish buttons. To redirect the user with one of these buttons, you use the `CancelDestinationPageUrl` or the `FinishDestinationPageUrl` attributes and set the appropriate URL as the destination.

Finally, you are not required to use the default text included with the buttons in the navigation system. You can change the text of each of the buttons with the use of the `CancelButtonText`, `FinishStepButtonText`, `FinishStepPreviousButtonText`, `NextStepButtonText`, `PreviousStepButtonText`, and the `StartStepNextButtonText` attributes.

Utilizing Wizard control events

One of the most convenient capabilities of the Wizard control is that it enables you to divide large forms into logical pieces. The end user can then work step by step through each section of the form. The developer, dealing with the inputted values of the form, has a few options because of the various events that are available in the Wizard control.

The Wizard control exposes events for each of the possible steps that an end user might take when working with the control. The following table describes each of the available events.

Event	Description
ActiveViewChanged	Triggers when the end user moves from one step to the next. It doesn't matter if the step is the middle or final step in the series. This event simply covers each step change generically.
CancelButtonClick	Triggers when the end user clicks the Cancel button in the navigation system.
FinishButtonClick	Triggers when the end user clicks the Finish button in the navigation system.
NextButtonClick	Triggers when the end user clicks the Next button in the navigation system.
PreviousButtonClick	Triggers when the end user clicks the Previous button in the navigation system.
SideBarButtonClick	Triggers when the end user clicks one of the links contained within the sidebar navigation of the Wizard control.

By working with these events, you can create a multisteped form that saves all the end users' input information when they change from one step to the next. You can also use the `FinishButtonClick` event and save everything that was stored in each of the steps at the end of the process. The nice thing about the Wizard control is that it remembers all the end user's input in each of the steps by means of the view state in the page. This enables you to work with all these values in the last step. This also gives the end user the capability to work back to previous steps and change values before these values are saved to a data store of some kind.

The event appears in your code-behind or inline code as shown in Listing 12-11.

Listing 12-11: The `FinishButtonClick` event

VB

```
<script runat="server">
    Sub Wizard1_FinishButtonClick(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.WizardNavigationEventArgs)

        End Sub
</script>
```

C#

```
<script runat="server">
    void Wizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
    {

    }
</script>
```

The main `<asp:Wizard>` element should also have the attribute shown in Listing 12-12 added to point at the new event.

Listing 12-12: The `<asp:Wizard>` element changes

```
<asp:Wizard ID="Wizard1" Runat="server" SideBarEnabled="true" ActiveStepIndex="0"
    OnFinishButtonClick="Wizard1_FinishButtonClick">
```

DynamicImage Server Control

Before the `DynamicImage` control came along, the only way to work with images in ASP.NET was to point to actual physical image files using either the `<asp:Image>` element or a `` HTML element. The physical images in ASP.NET 1.0/1.1 weren't always interpreted that well to other devices (especially smaller devices). It was also rather difficult and cumbersome to deal with images that were contained in a stream or received as a byte array.

ASP.NET 2.0 now includes a new `.axd` `HttpHandler` that is specifically developed for working with images not typically stored on disk.

Working with images from disk

Although the `DynamicImage` control is great for working with images that are in a stream, you can also use this control and simply point to images stored as a file on the server. This is illustrated in Listing 12-13.

Listing 12-13: Using the DynamicImage control with an image from disk

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>DynamicImage Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:DynamicImage ID="DynamicImage1" Runat="server"
      ImageFile="wrox_logo.gif">
    </asp:DynamicImage>
  </form>
</body>
</html>
```

The DynamicImage control uses the ImageFile attribute, which points to a Wrox logo that sits on a hard drive and displays this image in the browser. If you look at the properties of the picture in the browser, you see that the image displayed is exactly the same as the image stored on disk. The browser is quite capable of displaying .gif files without any difficulty. Without this control, if you try to pull open this page in a cell phone browser that understands only WML files, you have a problem. WML browsers do not understand .gif images and cannot display them. This isn't a problem for the new DynamicImage control because it automatically converts the image to a format that the consuming browser understands. Therefore, if you pull up the page from Listing 12-13 in a phone browser and compare it to the same image invoked in a typical browser, the two look like Figure 12-12.



Figure 12-12

In this example, the image shown in Microsoft Internet Explorer is a typical .gif image, whereas the image shown in the phone is a MIME-type image/vnd.wap.wbmp being automatically converted by the .axd HttpHandler. This feature alone makes it quite beneficial to work with the DynamicImage control.

Resizing images

Another outstanding feature of the new DynamicImage control is that, because it deals with the images as a stream, the control can perform modifications on the images before sending them onward to the container. For example, it can make larger images smaller.

As an example, I use a large image of my kids taken last winter that measures 1632 x 1224. This size is too large to display in the browser in a meaningful fashion. Therefore, I can use the DynamicImage control to manage the image's display size. Listing 12-14 shows how I can make the image smaller.

Listing 12-14: Resizing an image

```
<asp:DynamicImage ID="DynamicImage1" Runat="server"
    ImageFile="kids.jpg" Width="300">
</asp:DynamicImage>
```

To change the size of an image with the DynamicImage control, you use the Width or Height attributes of the <asp:DynamicImage> element. The image is automatically converted to the appropriate size. In this case, I use the Width attribute and give it a value of 300, which means that the width of the image is set to 300 pixels. No Height attribute is specified because the height of the image is changed along with the width of the image. The DynamicImage control constrains the proportions of the image in order to keep it set to a realistic factor. Remember that the large image is not actually compressed to a width of 300 pixels, but the image is redrawn so that it is only 300 pixels wide. This has huge ramifications, especially for the performance for your ASP.NET pages. My resized image is shown in Figure 12-13.

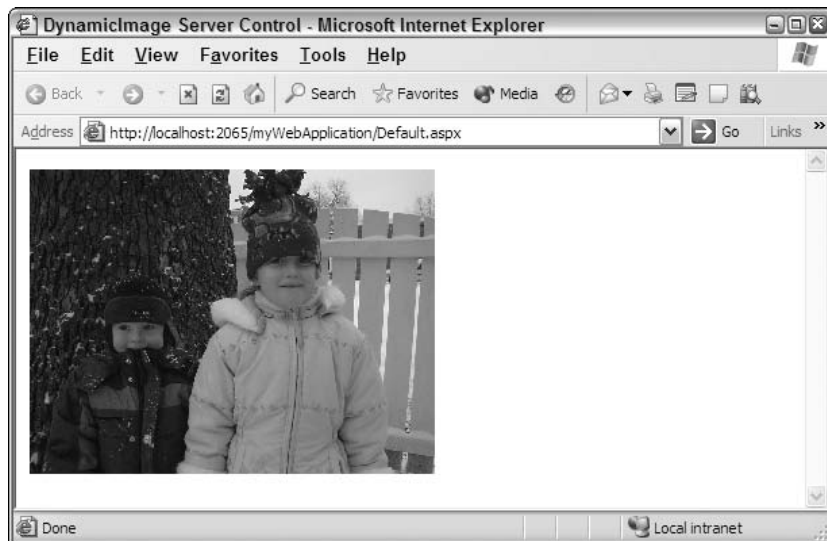


Figure 12-13

Now that you can resize images on the fly, you can see that it is relatively easy to create dynamic thumbnails of your images to display in the browser.

Displaying images from streams

In the Web world, you can utilize a lot of images in your ASP.NET pages, but they come at you in a non-typical format. For instance, if you retrieve an image sent from an XML Web service, invariably you receive this image as a byte array. Getting an image as a byte array was always a tricky procedure in ASP.NET 1.0/1.1. You had to save the file to disk before you could use it on your page. Now, with the use of the `DynamicImage` control, you can retrieve images that come in this format and place them directly on your page without saving them to disk first. You can also create dynamic images using technologies such as GDI+ and write these images directly to the browser. Listing 12-15 shows you how to use GDI+ to display a dynamically created image directly in the browser.

Listing 12-15: A GDI+ image displayed with a `DynamicImage` control

VB

```
<%@ Page Language="VB"%>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
    Public Function DrawText() As Bitmap
        Dim myBitmap As Bitmap = New Bitmap(300, 300)
        Dim myDraw As Graphics = Graphics.FromImage(myBitmap)
        myDraw.DrawString("This is actually an image.", _
            New Font("Arial", 12), Brushes.Black, 10, 10)
        myDraw.DrawLine(New Pen(Color.Black), 2, 2, 10, 10)
        myDraw.DrawLine(New Pen(Color.Black), 100, 100, 40, 40)

        Return myBitmap
    End Function

    Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        DynamicImage1.Image = DrawText()
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>GDI+</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:DynamicImage ID="DynamicImage1" Runat="server">
            </asp:DynamicImage>
        </form>
    </body>
</html>
```

C#

```

<%@ Page Language="C#" %>
<%@ Import Namespace="System.Drawing" %>

<script runat="server">
    public Bitmap DrawText()
    {
        Bitmap myBitMap = new Bitmap(300, 300);
        Graphics myDraw = Graphics.FromImage(myBitMap);
        myDraw.DrawString("This is actually an image.",
            new Font("Arial", 12), Brushes.Black, 10, 10);

        myDraw.DrawLine(new Pen(Color.Black), 2, 2, 10, 10);
        myDraw.DrawLine(new Pen(Color.Black), 100, 100, 40, 40);

        return myBitMap;
    }

    void Page_Load(object sender, EventArgs e)
    {
        DynamicImage1.Image = DrawText();
    }
</script>

```

From this example, you can see that a `DrawText` method performs some pretty non-artistic tasks in the creation of an image. In the `Page_Load` event, you use `DynamicImage` control's `Image` property and assign it the invocation of the `DrawText` method. As simply as that you create the image dynamically, which is then displayed in the browser (see Figure 12-14).



Figure 12-14

ImageMap Server Control

Another image control that is new to ASP.NET 2.0 is the ImageMap server control. This control enables you to turn an image into a navigation menu. In the past, many developers would break an image into multiple pieces and put them together again in a table, which reassembled the pieces into one image. In this way, when the end user clicks a particular piece of the overall image, the application can pick out which piece of the image was chosen and base actions upon that particular selection.

With the new ImageMap control, you can take a single image and specify particular hotspots of the image using coordinates. An example is shown in Listing 12-16.

Listing 12-16: Specifying sections of the image that are clickable

VB

```
<%@ Page Language="VB"%>

<script runat="server">
    Sub Imagemap1_Click(ByVal sender As Object, _
        ByVal e As System.Web.UI.WebControls.ImageMapEventArgs)

        Response.Write("You selected: " & e.PostBackValue)
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:ImageMap ID="Imagemap1" Runat="server" ImageUrl="kids.jpg"
            Width=300 OnClick="Imagemap1_Click" HotSpotMode="PostBack">
            <asp:RectangleHotSpot Top="0" Bottom="225" Left="0" Right="150"
                AlternateText="Henri" PostBackValue="Henri">
            </asp:RectangleHotSpot>
            <asp:RectangleHotSpot Top="0" Bottom="225" Left="151" Right="300"
                AlternateText="Sofia" PostBackValue="Sofia">
            </asp:RectangleHotSpot>
        </asp:ImageMap>
    </form>
</body>
</html>
```

C#

```
<%@ page language="C#"%>

<script runat="server">
    void Imagemap1_Click(object sender,
        System.Web.UI.WebControls.ImageMapEventArgs e) {

        Response.Write("You selected: " + e.PostBackValue);
    }
</script>
```

This page brings up an image of my two kids. If you click the left side of the image, you select Henri, and if you click the right side of the image, you select Sofia. You know which child you selected through a `Response.Write` statement, as shown in Figure 12-15.

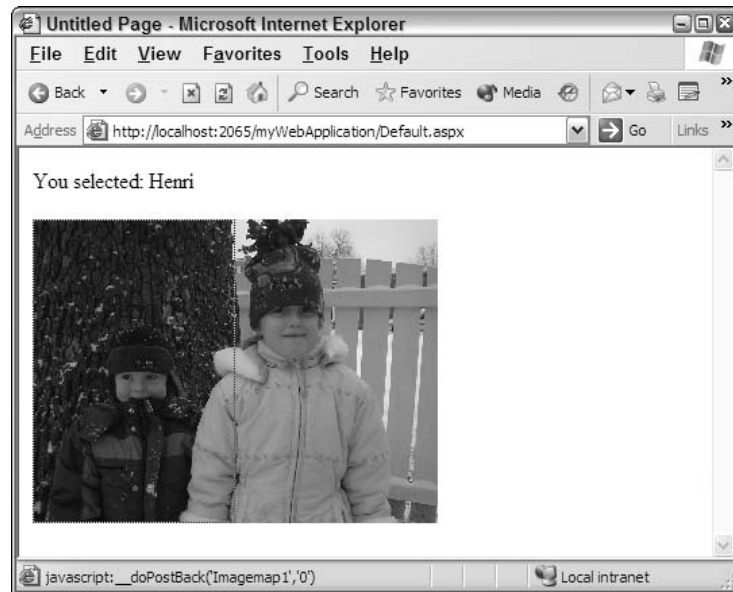


Figure 12-15

The `ImageMap` control enables you to specify hotspots in a couple of different ways. From the example in Listing 12-16, you can see that these hotspots are placed in a rectangular fashion using the `<asp:RectangleHotSpot>` element. This control takes the `Top`, `Bottom`, `Left`, and `Right` coordinates of the rectangle that is to be the hotspot. Besides the `<asp:RectangleHotSpot>` control, you can also use the `<asp:CircleHotSpot>` and the `<asp:PolygonHotSpot>` controls. Each control takes coordinates appropriate to its shape.

After you define the hotspots on the image, you can respond to the end user click of the hotspot in several ways. You first specify how you deal with the hotspot clicks in the root `<asp:ImageMap>` element with the use the `HotSpotMode` attribute.

The `HotSpotMode` attribute can take the values `PostBack`, `Navigate`, or `InActive`. In the previous example, the `HotSpotMode` value is set to `PostBack` — meaning that after the end user clicks the hotspot, you want to postback to the server and deal with the click at that point.

Because the `HotSpotMode` is set to `PostBack` and you have created several hotspots, you must determine which hotspot is selected. You make this determination by giving each hotspot (`<asp:RectangleHotSpot>`) a postback value with the `PostBackValue` attribute. In the first hotspot, the example uses `Henri` as the value, whereas the second hotspot uses `Sofia` as the value.

The `PostBackValue` attribute is also the helper text that appears in the browser (in the yellow box) directly below the mouse cursor when the end user hovers the mouse over the hotspot.

Chapter 12

After the end user clicks one of the hotspots from the previous example, the event procedure displays the value that was selected in a `Response.Write` statement.

Instead of posting back to the server, you can also navigate to an entirely different URL when a particular hotspot is selected. To accomplish this, you change the `HotSpotMode` attribute in the main `<asp:ImageMap>` element to the value `Navigate`. Then within the `<asp:RectangleHotSpot>` elements, simply use the `NavigateUrl` attribute and assign the location to which the end user should be directed if this particular hotspot is clicked:

```
<asp:ImageMap ID="Imagemap1" Runat="server" ImageUrl="kids.jpg"
HotSpotMode="Navigate">
  <asp:RectangleHotSpot Top="0" Bottom="225" Left="0" Right="150"
  AlternateText="Henri" NavigateUrl="HenriPage.aspx">
</asp:RectangleHotSpot>
  <asp:RectangleHotSpot Top="0" Bottom="225" Left="151" Right="300"
  AlternateText="Sofia" NavigateUrl="SofiaPage.aspx">
</asp:RectangleHotSpot>
</asp:ImageMap>
```

Summary

New server controls are fun. When they were first introduced, I kept saying to myself “this will save me a ton of time,” “this will be so useful,” “this is going in my next project,” and so on. This chapter introduced you to some of these new controls and to the different ways you might incorporate them into your next projects.

The `BulletedList` control enables you to create all sorts of bulleted lists either directly from inline items or from items contained in a data store of some kind. The `HiddenField` control allows for server-side access to a very important HTML element that was formerly far more difficult to work with. Other controls covered include the `FileUpload` control, which enables you to upload files easily to the server. I also covered the `MultiView` and `View` controls for working through processes, the `Wizard` control for advanced process work, the `DynamicImage` control for working with images not found on disk, and finally the `ImageMap` control for creating hotspots on an image. All these controls are wonderful options to use on any of your ASP.NET pages, making it much easier to develop the functionality that your pages require.

13

Changes to ASP.NET 1.0 Controls

This book, thus far, has given you a close look at the substantial new features in ASP.NET version 2.0, but you haven't yet seen the changes that have taken place to the core controls already present in ASP.NET 1.0/1.1. These controls still work as they did before. ASP.NET 2.0 is backward compatible with the previous two versions of ASP.NET. This means that the control code that you wrote in these past versions works in ASP.NET 2.0, but now you have more functionality for some of these controls.

This chapter focuses on the enhancements to these controls that make them even better today. Some of the improvements are minor, but others are quite dramatic. Some of the changes to the simpler controls are reviewed first.

Label Server Control

The Label server control has always been a control that simply showed text. Now, it has a little bit of extra functionality. The big change to the Label server control is that you can now give items in your form *hotkey* functionality (also known as *accelerator* keys). This causes the page to focus on a particular server control that you declaratively assign to a specific hotkey press (for example, using Alt+N to focus on the first text box on the form).

A hotkey is a quick way for the end user to initiate an action on the page. For instance, with Microsoft Internet Explorer you can press Ctrl+N to open a new instance of IE. Hotkeys have always been quite common in thick-client applications (Windows Forms), so you can now use them in ASP.NET. Listing 13-1 shows an example of how to give hotkey functionality to two text boxes on a form.

Listing 13-1: Using the Label server control to provide hotkey functionality

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Label Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <p>
            <asp:Label ID="Label1" Runat="server" AccessKey="N"
                AssociatedControlId="Textbox1">User<u>n</u>ame</asp:Label>
            <asp:Textbox ID="Textbox1" Runat="server"></asp:Textbox></p>
        <p>
            <asp:Label ID="Label2" Runat="server" AccessKey="P"
                AssociatedControlId="Textbox2"><u>P</u>assword</asp:Label>
            <asp:Textbox ID="Textbox2" Runat="server"></asp:Textbox></p>
        <p>
            <asp:Button ID="Button1" Runat="server" Text="Submit" />
        </p>
    </form>
</body>
</html>
```

Hotkeys are assigned with the `AccessKey` attribute. In this case, `Label1` uses `N`, while `Label2` uses `P`. The second new attribute for the Label control is the `AssociatedControlId` attribute. The `String` value placed here associates the Label control to another server control on the form. The value must be one of the other server controls on the form. If not, the page gives you an error when invoked.

Now that these two controls are in place, when the page is called in the browser, you can press `Alt+N` or `Alt+P` to automatically focus on a particular text box in the form. Notice in Figure 13-1 that I placed HTML declared underlines for the letter to be pressed along with the `Alt` key that creates focus on the control adjoining the text. This is not required, but I highly recommend it because it is what the end user expects when working with hotkeys. In this case, I have underlined the letter `n` in `Username` and the letter `P` in `Password`.

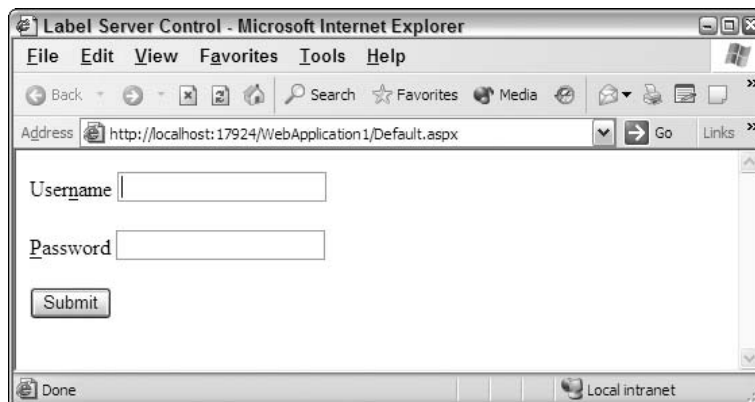


Figure 13-1

When working with hotkeys, be aware that not all letters are available to use with the Alt key. Microsoft Internet Explorer already uses Alt+F, E, V, I, O, T, A, W, and H. If you use any of these letters, IE actions supersede any actions on the page.

Button, LinkButton, and ImageButton Server Controls

Frequently, buttons are used for submitting information and causing actions to occur on a Web page. Before ASP.NET 1.0/1.1, people intermingled quite a bit of JavaScript in their pages to fire JavaScript events when a button was clicked. This process became more cumbersome in ASP.NET 1.0/1.1, but now with ASP.NET 2.0, it is much easier.

You can create a page that has a JavaScript event, as well as a server-side event, triggered when the button is clicked, as illustrated in Listing 13-2.

Listing 13-2: Two types of events for the button

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Response.Write("Postback!")
    End Sub
</script>

<script language=javascript>
    function AlertHello()
    {
        alert('Hello ASP.NET');
    }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Button Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Button ID="Button1" Runat="server" Text="Button"
            OnClientClick="AlertHello()" OnClick="Button1_Click" />
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void Button1_Click(object sender, EventArgs e)
```

(continued)

Listing 13-2: (continued)

```
{  
    Response.Write("Postback!");  
}  
</script>
```

A couple of things are happening here. The first thing to notice is the new attribute for the Button server control: `OnClientClick`. This attribute points to the client-side function, unlike the `OnClick` attribute that points to the server-side event. In this case, I have created a JavaScript function called `AlertHello()`.

One cool thing about Visual Studio 2005 is that now it can work with server-side script tags that are right alongside client-side script tags. This all works together seamlessly now. In the example, after the JavaScript alert dialog is issued (see Figure 13-2) and the end user clicks OK, the page posts back as the server-side event is triggered.



Figure 13-2

Another new and exciting attribute for the button controls is the `PostBackUrl` attribute. It enables you to perform cross-page posting, instead of simply posting your ASP.NET pages back to the same page, as shown in the following example:

```
<asp:Button ID="Button2" Runat="server" Text="Submit page to Page2.aspx"  
    PostBackUrl="Page2.aspx" />
```

Cross-page posting is covered in greater detail in Chapter 3.

DropDownList, ListBox, CheckBoxList, and RadioButtonList Server Controls

The `DropDownList`, `ListBox`, `CheckBoxList`, and `RadioButtonList` server controls now give you the capability to visually remove items from the collection displayed in the control, although you can still work with the items that aren't displayed in your server-side code. For a quick example of this, create a drop-down

list with three items, one of which will not display. On the postback, however, you should still be able to work with the ListItem's Value or Text property, as illustrated in Listing 13-3.

Listing 13-3: Disabling certain ListItems from a collection

VB

```
<%@ page language="VB" %>

<script runat="server">
    Sub DropDownList1_SelectedIndexChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        Response.Write("You selected item number " & _
            DropDownList1.SelectedValue & "<br>")
        Response.Write("You didn't select item number " & _
            DropDownList1.Items(1).Value)
    End Sub
</script>

<html>
<head runat="server">
    <title>DropDownList Server Control</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:DropDownList ID="DropDownlist1" Runat="server" AutoPostBack="True"
            OnSelectedIndexChanged="DropDownlist1_SelectedIndexChanged">
            <asp:ListItem Value="1">First Choice</asp:ListItem>
            <asp:ListItem Value="2" Enabled="False">Second Choice</asp:ListItem>
            <asp:ListItem Value="3">Third Choice</asp:ListItem>
        </asp:DropDownList>
    </form>
</body>
</html>
```

C#

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

<script runat="server">
    void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        Response.Write("You selected item number " +
            DropDownList1.SelectedValue + "<br>");
        Response.Write("You didn't select item number " +
            DropDownList1.Items(1).Value);
    }
</script>
```

From the code, you can see that the `<asp:listitem>` element has a new attribute: `Enabled`. The Boolean value given to this element dictates whether an item in the collection is displayed. If you use `Enabled="False"`, the item is not displayed, but you still have the capability to work with the item if required in the server-side code displayed in the `DropDownList1_SelectedIndexChanged` event. The result of the output from these `Response.Write` statements is shown in Figure 13-3.

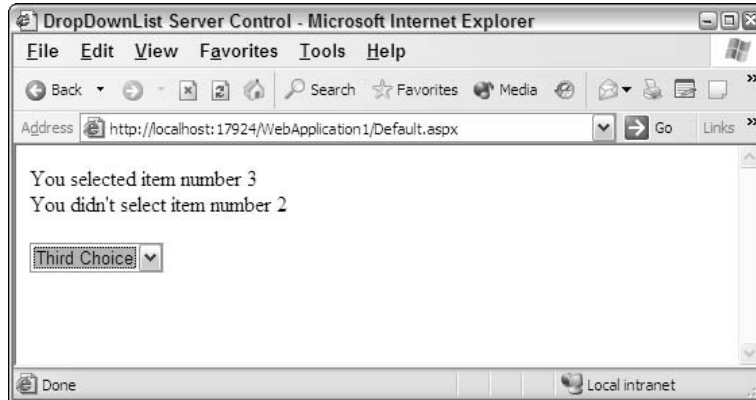


Figure 13-3

Image Server Control

Special circumstances can prevent end users from viewing an image that is part of your Web page. They might be physically unable to see the image, or they might be using a text-only browser. In these cases, their browsers look for the `` element's `longdesc` attribute, which points to a file containing a long description of the image that is displayed.

For these cases, the Image server control now includes a new `DescriptionUrl` attribute. The value assigned to this attribute is a text file, which contains a thorough description of the image with which it is associated. You use it as shown in Listing 13-4.

Listing 13-4: Using the `DescriptionUrl` attribute

```
<asp:Image ID="Image1" Runat="server" DescriptionUrl="~/Image01.txt" />
```

This produces the following results in the browser:

```

```

Table Server Control

The Table server control is enhanced with some extra features as well. One of the simpler new additions is the capability to add captions to the tables on Web pages. A table with a caption is displayed in Figure 13-4.

To give your page a caption, simply use the new `Caption` attribute in the Table control as illustrated in Listing 13-5.

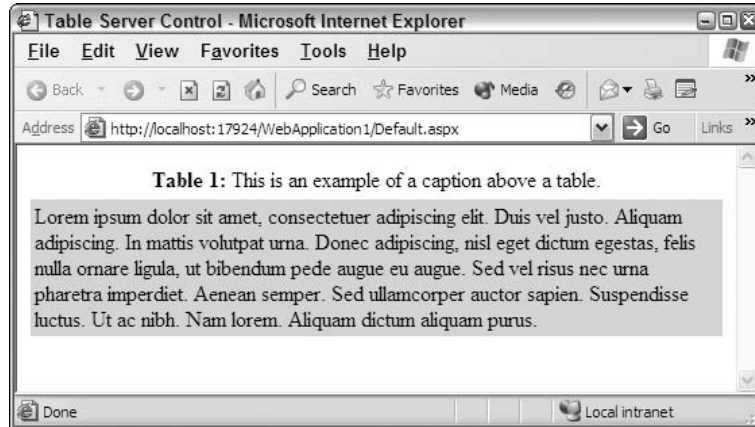


Figure 13-4

Listing 13-5: Using the new Caption attribute

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" ><head runat="server">
  <title>Table Server Control</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:Table ID="Table1" Runat="server"
      Caption="<b>Table 1:</b> This is an example of a caption above a table."
      BackColor=Gainsboro>
      <asp:TableRow ID="Tablerow1" Runat=server>
        <asp:TableCell ID="Tablecell1" Runat="server">Lorem ipsum dolor sit
          amet, consectetur adipiscing elit. Duis vel justo. Aliquam
          adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
          dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
          eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
          semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
          nibh. Nam lorem. Aliquam dictum aliquam purus.</asp:TableCell>
        </asp:TableRow>
      </asp:Table>
    </form>
  </body>
</html>
```

By default, the caption is placed at the top center of the table, but you can also control where the caption is placed by using the second new attribute — `CaptionAlign`. The possible settings for this attribute include `Bottom`, `Left`, `NotSet`, `Right`, and `Top`.

In the past, an `<asp:Table>` element contained any number of `<asp:TableRow>` elements. Now you have some additional elements that can be nested within the `<asp:Table>` element. These new elements include the `<asp:TableHeaderRow>` and the `<asp:TableFooterRow>` elements. These two new elements add either a header or footer to your table. These headers and footers enable you to use the Table server control to page through lots of data but still retain some text in place to indicate the type of data being handled. This is quite powerful when you work with mobile applications because sometimes end users are able to move through only a few records at a time.

Literal Server Control

The Literal server control was always used in the past for text that you wanted to remain unchanged on your page. The Label server control altered the output by placing `` elements around the text as shown:

```
<span id="Label1">Here is some text</span>
```

The Literal control just output the text without the `` elements. The Literal server control now includes the new attribute `Mode` that allows you to dictate how the text assigned to the control is interpreted by the ASP.NET engine.

If you place some HTML code in the string that is output (for instance, `Here is some text`), the Literal control outputs just that and the consuming browser shows the text as bold:

```
Here is some text
```

Adding `Mode="Encode"` encodes the output before it is received by the consuming application:

```
&lt;b&gt;Label&lt;/b&gt;
```

Now, instead of the text being converted to a bold font, the `` elements are displayed:

```
<b>Here is some text</b>
```

This is ideal if you want to display code in your application. Other values for the `Mode` attribute include `Transform` and `PassThrough`. `Transform` looks at the consumer and includes or removes elements as needed. For instance, not all devices accept HTML elements and, if the value of the `Mode` attribute is set to `Transform`, these elements are removed from the string before it is sent to the consuming application. A value of `PassThrough` for the `Mode` property means that the text is sent to the consuming application without any changes made to the string.

AdRotator Server Control

Although Web users find ads rather annoying, advertising continues to be prevalent everywhere on the Web. The AdRotator server control has been enhanced quite a bit to give you several different ways to incorporate ads into your Web applications. The biggest change to this control is that it now enables you

to utilize pop-up or pop-under ads in addition to the standard banner ads. With the AdRotator control you can now use advertisement data from sources other than the standard XML file that was used with the previous versions of this control.

If using an XML source for the ad information, you first create an XML advertisement file. This advertisement file is quite similar to the previous advertisement file, but you can now incorporate some new elements that give you even more control over the appearance and behavior of your ads. Listing 13-6 shows an example of the XML advertisement file.

Listing 13-6: The XML advertisement file

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <ImageUrl>book1.gif</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Visit Wrox Today!</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>VB.NET</Keyword>
    <Height>126</Height>
    <Width>100</Width>
  </Ad>
  <Ad>
    <ImageUrl>book2.gif</ImageUrl>
    <NavigateUrl>http://www.wrox.com</NavigateUrl>
    <AlternateText>Visit Wrox Today!</AlternateText>
    <Impressions>50</Impressions>
    <Keyword>XML</Keyword>
    <Height>125</Height>
    <Width>100</Width>
  </Ad>
</Advertisements>
```

This XML file, used for storing information about the advertisements that appear in your application, has some new elements detailed in the following table.

New Element	Description
CounterGroup	Specifies the group that is used by the new site counter capabilities in ASP.NET.
CounterName	Specifies the name of the counter used.
Height	Takes a numerical value that indicates the height of the ad in pixels.
Width	Takes a numerical value that indicates the width of the ad in pixels.

Now that the XML advertisement file is in place, you can simply use the AdRotator control as before. This is shown in Listing 13-7.

Listing 13-7: Using the AdRotator control as a banner ad

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>AdRotator Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:AdRotator ID="AdRotator1" Runat="server"
            AdvertisementFile="MyAds.xml" />
        <p>Lorem ipsum dolor sit
            amet, consectetur adipiscing elit. Duis vel justo. Aliquam
            adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
            dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
            eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
            semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
            nibh. Nam lorem. Aliquam dictum aliquam purus.</p>
    </form>
</body>
</html>
```

This example shows the ad that is specified in the XML advertisement file as a banner ad at the top of the page. You can easily change the AdRotator control, however, so that instead of appearing directly on the page, it displays as a pop-up or pop-under ad instead. A pop-up ad is an ad that is shown in a separate browser instance on top of the page that is currently being viewed, whereas a pop-under ad is an ad that is shown in a separate browser instance that appears under the page that is currently being viewed. In the latter case, the end user might not see the ad until much later in the browsing experience.

To change the ad from Listing 13-7 so that it appears as a pop-up, you use the AdType property:

```
<asp:AdRotator ID="AdRotator1" Runat="server"
    AdvertisementFile="MyAds.xml" AdType="popup" />
```

This causes the ad to appear as a pop-up ad as illustrated in Figure 13-5.

To have the ad appear as a pop-under ad, change the value of the AdType property to PopUnder:

```
<asp:AdRotator ID="AdRotator1" Runat="server"
    AdvertisementFile="MyAds.xml" AdType="popunder" />
```

As you can see from Figure 13-5, the ad appears as a pop-up directly in the center of the screen. This is the default behavior (also for the pop-under ads). To change the location of the pop-up or pop-under ads on the screen, you can use the AdRotator's PopPositionLeft and PopPositionTop attributes:

```
<asp:AdRotator ID="AdRotator1" Runat="server"
    PopPositionLeft="25" PopPositionTop="25"
    AdvertisementFile="MyAds.xml" AdType="popup" />
```



Figure 13-5

This change causes the ad to appear 25 pixels down from the top-left corner of the screen and 25 pixels over from the left side of the screen, as illustrated in Figure 13-6.



Figure 13-6

You can also control the frequency of the pop-up or pop-under rate with the new `PopFrequency` attribute to ensure these pop-ups or pop-unders don't appear each and every time someone visits the page:

```
<asp:AdRotator ID="AdRotator1" Runat="server"
  PopFrequency="50"
  AdvertisementFile="MyAds.xml" AdType="popup" />
```

The default value of the `PopFrequency` attribute is 100, which means that the pop-up or pop-under ads appear 100% of the time. Changing the value to 50, as shown in the preceding code snippet, means that the ads appear in only 50% of the instances when someone invokes the page in the browser.

You are not required to place all your ad information in the XML advertisement file, but instead, you can use another data source to which you bind the AdRotator. For instance, you bind the AdRotator to a `SqlDataSource` object that is retrieving the ad information from SQL Server in the following fashion:

```
<asp:AdRotator ID="AdRotator1" Runat="server"
  DataSourceId="SqlDataSource1" AlternateTextField="AlternateTF"
  ImageUrlField="Image" NavigateUrlField="NavigateUrl"
  AdType="popup" />
```

The `AlternateTextField`, `ImageUrlField`, and `NavigateUrlField` properties point to the column names that are used in SQL Server for those items.

Panel Server Control

The Panel server control encapsulates a set of controls you can use to manipulate or lay out your ASP.NET pages. The Panel control is basically a wrapper for other controls. It enables you to take a group of server controls along with other elements (such as HTML and images) and turn them into a single unit.

The advantage of using the Panel control to encapsulate a set of other elements is that as a single unit, you can manipulate these elements with a single attribute set in the Panel control itself. For example, setting the `Font-Bold` attribute to `True` causes each item within the panel control to adopt this attribute.

The new addition to the Panel control is the capability to scroll with scrollbars that appear automatically depending on the amount of information that Panel control holds. You can even specify how the scrollbars should appear.

For an example of using scrollbars, look at a long version of the Lorem Ipsum text (found at www.lipsum.com) and place that text within the Panel control as shown in Listing 13-8.

Listing 13-8: Using the new scrollbar feature with the Panel server control

```
<%@ Page Language="VB" %>

<html>
<head runat="server">
  <title>Panel Server Control Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <asp:Panel ID="Panel1" Runat="server" Height="300" Width="300"
      ScrollBars="auto">
      <p>Lorem ipsum dolor sit amet...</p>
    </asp:Panel>
  </form>
</body>
</html>
```

By assigning values to the `Height` and `Width` attributes of the `Panel` server control and using the `ScrollBars` attribute (in this case, set to `Auto`), you can display the information it contains within the defined area using scrollbars (see Figure 13-7).

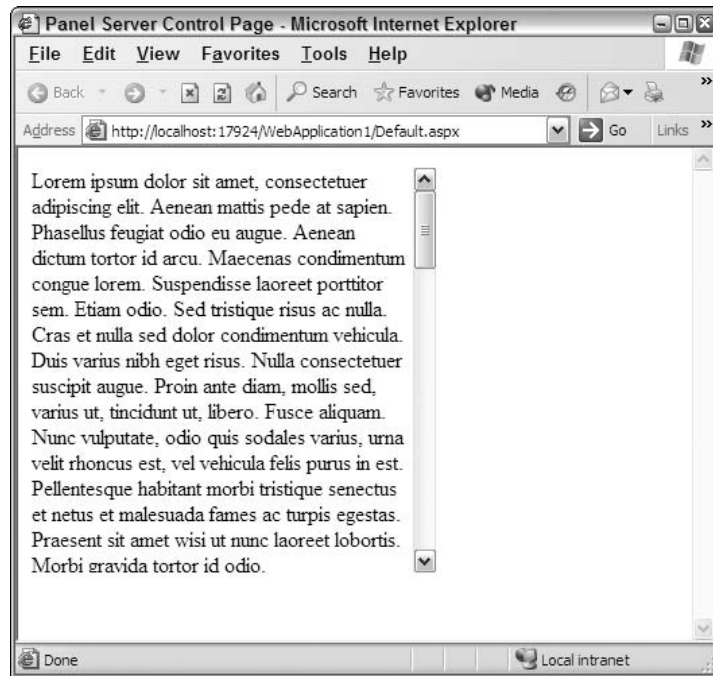


Figure 13-7

As you can see from the figure, a single vertical scrollbar has been added to the set area of 300 x 300 pixels. The `Panel` control wraps the text by default as required. To change this behavior, you can use the new `Wrap` attribute, which takes a Boolean value:

```
<asp:Panel ID="Panel1" Runat="server"
    Height="300" Width="300" ScrollBars="Auto"
    Wrap="False" />
```

Turning off wrapping may cause the horizontal scrollbar to turn on (depending on what is contained in the panel section).

If you don't want to let the ASP.NET engine decide which scrollbars to activate, you can actually make that decision with the use of the `ScrollBars` attribute. Other values of this attribute besides `Auto` include `None`, `Horizontal`, `Vertical`, and `Both`.

Another interesting attribute that allows you to change the behavior of the `Panel` control is the `HorizontalAlign` attribute. When using this attribute, you can set how the content in the `Panel` control is horizontally aligned. The possible values of this attribute include `NotSet`, `Center`, `Justify`, `Left`, and `Right`. Figure 13-8 shows a collection of `Panel` controls with different horizontal alignments for each.

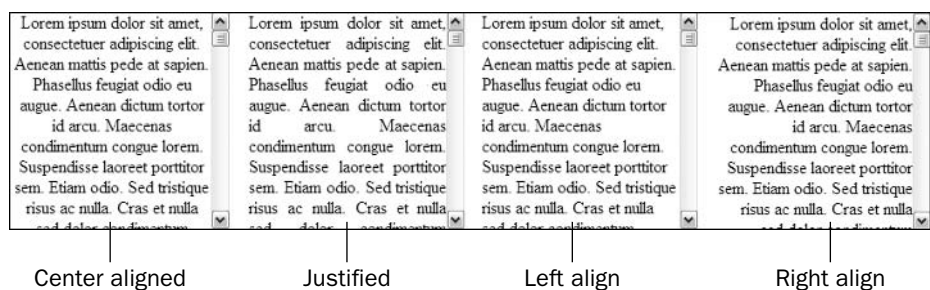


Figure 13-8

It is also possible to move the vertical scrollbar to the left side of the Panel control with the use of the `Direction` attribute. This attribute can be set to `NotSet`, `LeftToRight`, and `RightToLeft`. A setting of `RightToLeft` is ideal when you are dealing with languages that are written from right to left (some Asian languages, for example). This setting, however, also moves the scrollbar to the left side of the Panel control. If the scrollbar is moved to the left side and the `HorizontalAlign` attribute is set to `Left`, your content resembles Figure 13-9.

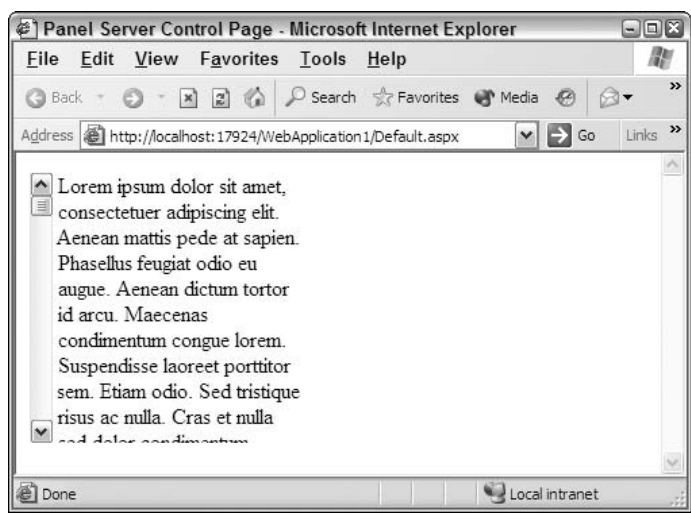


Figure 13-9

Validation Server Controls

In many instances, developers want to place more than one form on a single page. This was always possible in ASP.NET 1.0/1.1 because different button clicks could be used to perform different server-side events. Some issues related to this type of construction were problematic, however.

One of these issues was the difficulty of having validation controls for each of the forms on the page. Different validation controls were often assigned to two distinct forms on the page. When the end user

submitted one form, the validation controls in the other form were fired (because the user was not working with that form), thereby stopping the first form from being submitted.

Figure 13-10, for example, shows a basic page for the St. Louis .NET User Group that includes two forms.



Figure 13-10

One of the forms is for members of the site to supply their usernames and passwords to log in to the Members' Only section of the site. The second form on the page is for anyone who wishes to sign up for the user group's newsletter. Each form has its own button and some validation controls associated with it. The problem arises when someone submits information for one of the forms. For instance, if you are a member of the group, supply your username and password, and click the Login button, the validation controls for the newsletter form fire because no e-mail address was placed in that particular form. If someone interested in getting the newsletter places an e-mail address in the last text box and clicks the Sign-up button, the validation controls in the first form fire because no username and password were input in that form.

ASP.NET 2.0 now provides you with a `ValidationGroup` property that enables you to separate the validation controls into separate groups. It allows you to activate only the required validation controls when an end user clicks a button on the page. Listing 13-9 shows an example of separating the validation controls on a user group page into different buckets.

Listing 13-9: Using the `ValidationGroup` property

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Validation Groups</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>St. Louis .NET User Group</h1>
```

(continued)

Listing 13-9: (continued)

```

<p>Username:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>&nbsp; Password:
<asp:TextBox ID="TextBox2" Runat="server"
    TextMode="Password"></asp:TextBox>&nbsp;
<asp:Button ID="Button1" Runat="server" Text="Login"
    ValidationGroup="Login" />
    <br />
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
        ErrorMessage="* You must submit a username!"
        ControlToValidate="TextBox1" ValidationGroup="Login">
    </asp:RequiredFieldValidator>
    <br />
    <asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
        ErrorMessage="* You must submit a password!"
        ControlToValidate="TextBox2" ValidationGroup="Login">
    </asp:RequiredFieldValidator>
</p>
    Our main meeting is almost always held on the last Monday of the month.
    Sometimes due to holidays or other extreme circumstances,
    we move it to another night but that is very rare. Check the home page
    of the web site for details. The special
    interest groups meet at other times during the month. Check the SIG
    page and visit their individual sites for more information.
    You can also check out calendar page for a summary of events.<br />
</p>
<h2>Sign-up for the newsletter!</h2>
<p>Email:
<asp:TextBox ID="TextBox3" Runat="server"></asp:TextBox>&nbsp;
<asp:Button ID="Button2" Runat="server" Text="Sign-up"
    ValidationGroup="Newsletter" />&nbsp;
    <br />
    <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
        Runat="server"
        ErrorMessage="* You must submit a correctly formatted email address!"
        ControlToValidate="TextBox3" ValidationGroup="Newsletter">
    </asp:RegularExpressionValidator>
    <br />
    <asp:RequiredFieldValidator ID="RequiredFieldValidator3" Runat="server"
        ErrorMessage="* You forgot your email address!"
        ControlToValidate="TextBox3" ValidationGroup="Newsletter">
    </asp:RequiredFieldValidator>
</p>
</form>
</body>
</html>

```

The use of the `ValidationGroup` property in this page is shown in bold. You can see that this property takes a `String` value. The other item to notice is that not only validation controls have this new property. The core server controls also have the `ValidationGroup` property because things like button clicks must be associated with specific validation groups.

In this example, each of the buttons has a distinct validation group assignment. The first button on the form uses `Login` as a value and the second button on the form uses `Newsletter` as a value. Then each of the validation controls is associated with one of these validation groups. Because of this, when the end user clicks on the Login button on the page, ASP.NET recognizes that it should work only with the validation server controls that have the same validation group name. ASP.NET ignores the other validation controls that are assigned to other validation groups.

Using this enhancement, you can now have multiple sets of validation rules fired only when you want them to be fired (see Figure 13-11).



Figure 13-11

Another great feature that has been added to validation controls is a property called `SetFocusOnError`. This property takes a `Boolean` value and, if a validation error is thrown when the form is submitted, the property places the page focus on the form element that receives the error. The `SetFocusOnError` property is used in the following example:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
    ErrorMessage="* You must submit a username!"
    ControlToValidate="TextBox1" ValidationGroup="Login" SetFocusOnError="True">
</asp:RequiredFieldValidator>
```

If `RequiredFieldValidator1` throws an error because the end user didn't place a value in `TextBox1`, the page is redrawn with the focus in `TextBox1`, as shown in Figure 13-12.

Note that if you have multiple validation controls on your page with the `SetFocusOnError` property set to `True` and there is more than one validation error, the uppermost form element that has a validation error gets the focus. In the previous example, if both the username text box (`TextBox1`) and the password text box (`TextBox2`) have validation errors associated with them, the page focus is assigned to the username text box because it is the first control on the form with an error.



Figure 13-12

Summary

In this chapter, I showed you the numerous new capabilities and features added to ASP.NET in this latest version, and they are pretty astounding. You have more than 40 new server controls at your disposal, besides some great changes to the server controls that you already use on a day-to-day basis from ASP.NET 1.0/1.1.

This chapter covered changes to controls such as the Label, Table, and Validation server controls. The new features added to classic server controls are, in many ways, just as outstanding as the new controls that appear in ASP.NET 2.0. The new features added to the classic controls make it easy to extend the capabilities of your ASP.NET applications.

14

Administration and Management

The exciting 2.0 version of ASP.NET adds a lot of punch to developer productivity. In addition to incorporating a lot of common Web site functionality, new features in ASP.NET make it easier to manage ASP.NET applications. These new ASP.NET administration and management capabilities are designed to fill the pressing needs of developers. This chapter introduces the new management dialogs for working with ASP.NET applications that run on IIS, as well as showing you the new ASP.NET Web Site Administration Tool.

The MMC ASP.NET Snap-In

One great new addition to ASP.NET management is the ASP.NET tab in the Microsoft Management Console if you are using IIS for the basis of your ASP.NET applications. To get at this new ASP.NET tab, open IIS and expand the `Web Sites` folder. This folder shows a list of all the Web sites configured to work with IIS. Remember that not all of your Web sites are configured to work in this manner. It is also possible to create ASP.NET applications that make use of the new ASP.NET built-in Web server.

After you find the application you are looking for in the `Web Sites` folder, right-click that application and select `Properties` (see Figure 14-1).

Selecting the `Properties` option brings up the MMC console. The far-right tab is the ASP.NET tab. Click this tab to get the results shown in Figure 14-2. You should also note that selecting one of the application folders lets you edit the `web.config` file from the MMC snap-in; selecting `Properties` for the default Web site (the root node) lets you edit the `machine.config` file.

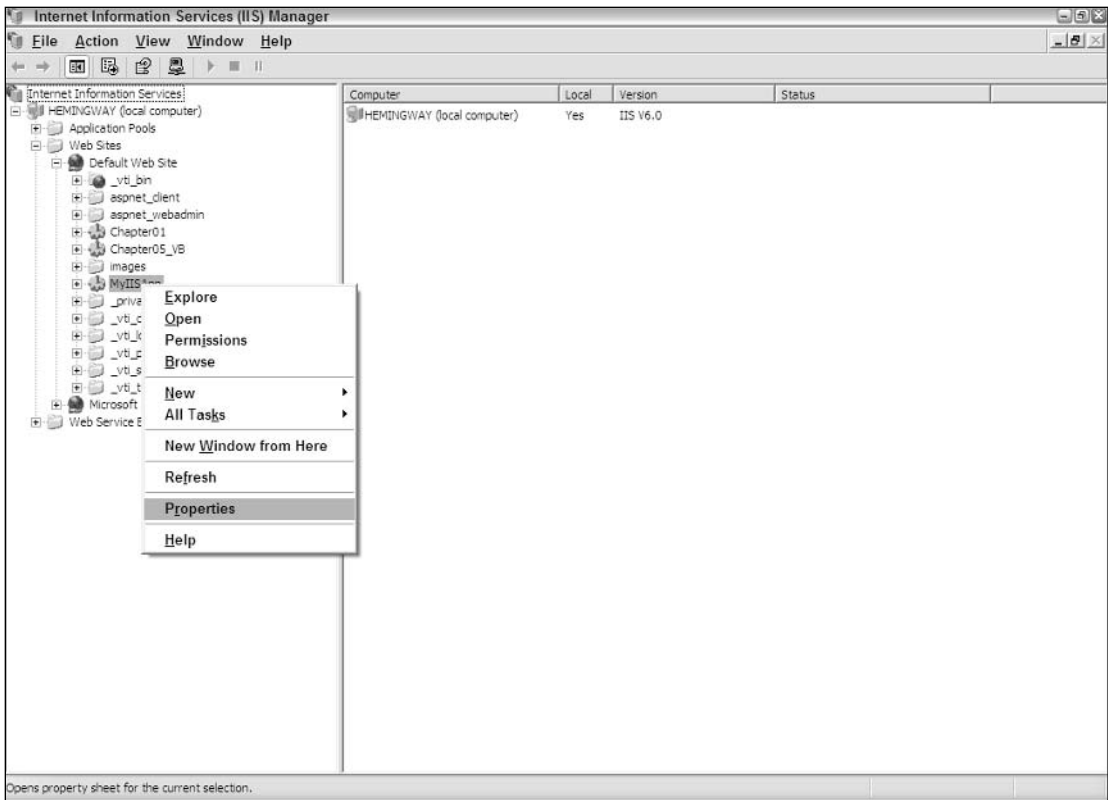


Figure 14-1

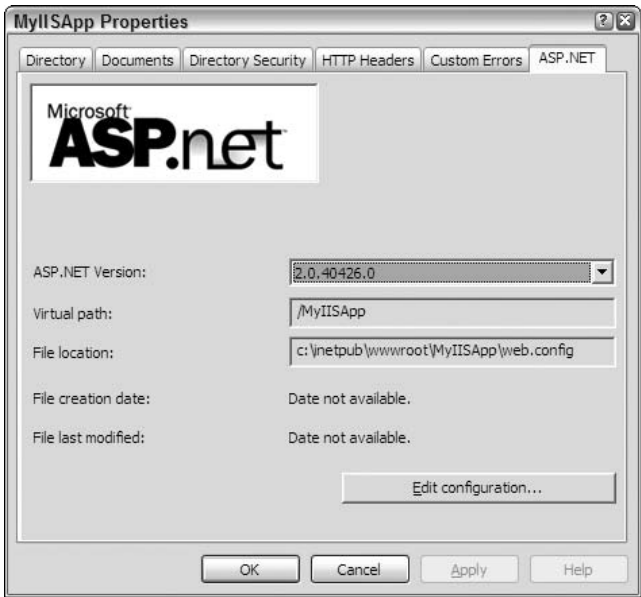


Figure 14-2

The top configuration panel for ASP.NET enables you to change the following items:

- ❑ **ASP.NET Version:** The .NET Framework version number on which the ASP.NET application is to run. Be careful about switching versions of the application. Some minor breaking changes may cause errors in different versions of the framework.
- ❑ **Virtual path:** The virtual path of the application. In this case, Visual Studio creates an application titled `MyIISApp` with a `MyIISApp` virtual directory.
- ❑ **File location:** The location of the file being altered by the MMC console. In most cases, these configuration GUIs alter the `web.config` file. In this case, the file location is the `web.config` file in the `MyIISApp` application.
- ❑ **File creation date:** The date when the `web.config` file was created.
- ❑ **File last modified:** The date when the `web.config` file was last modified either manually, using the MMC console, or by the ASP.NET Web Site Administration Tool.

In addition to these items, the ASP.NET tab also includes an Edit Configuration button that provides a tremendous amount of modification capabilities to use in the `web.config` file. When you click this button, you see a multitabbed GUI titled ASP.NET Configuration Settings. The following sections review each of the tabs available to you through this MMC console.

General

The first tab, labeled General, enables you to manage connection strings and app settings for your application. Figure 14-3 shows an example of one setting for an application.

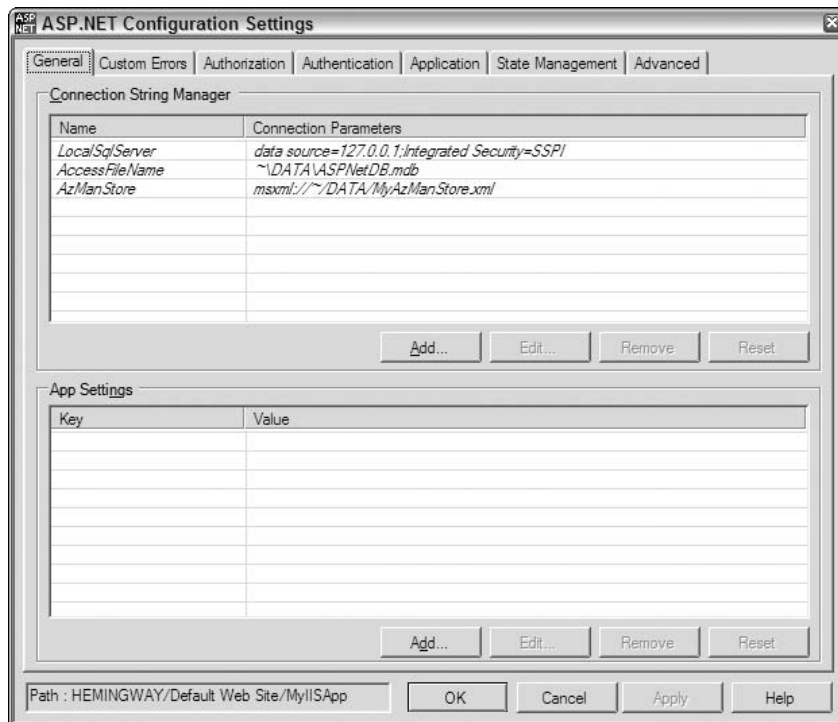


Figure 14-3

Chapter 14

The General tab has two sections. One is for adding, editing, or removing connection strings; the other is for adding, editing, and removing app settings. Both of these items work with name/value pairs. If you choose to add a connection string to your application, click the Add button in the top section (Connection String Manager) to see the dialog in Figure 14-4.

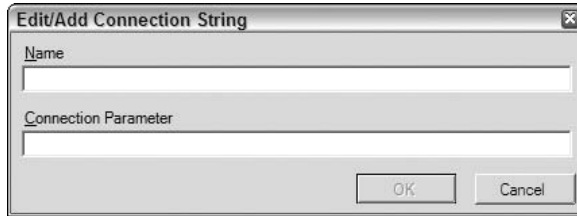


Figure 14-4

The Edit/Add Connection String dialog asks for the Name and the Connection Parameter for the connection string. Supplying this information and clicking OK provides your application with a connection string.

If you select the Edit/Add buttons in the bottom section (App Settings), you see the dialog shown in Figure 14-5.

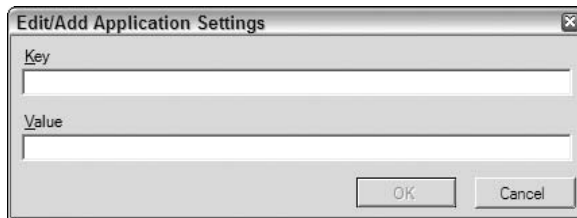


Figure 14-5

The Edit/Add Application Settings dialog asks for a Key and Value pair. After you add these items and click OK, the settings appear in the list in the main dialog. You can now either edit or delete the settings from the application.

Custom Errors

The second tab is the Custom Errors tab. This section of the console enables you to add custom error pages or redirect users to particular pages when a specific error occurs in the application. Figure 14-6 shows an example of the Custom Errors tab.

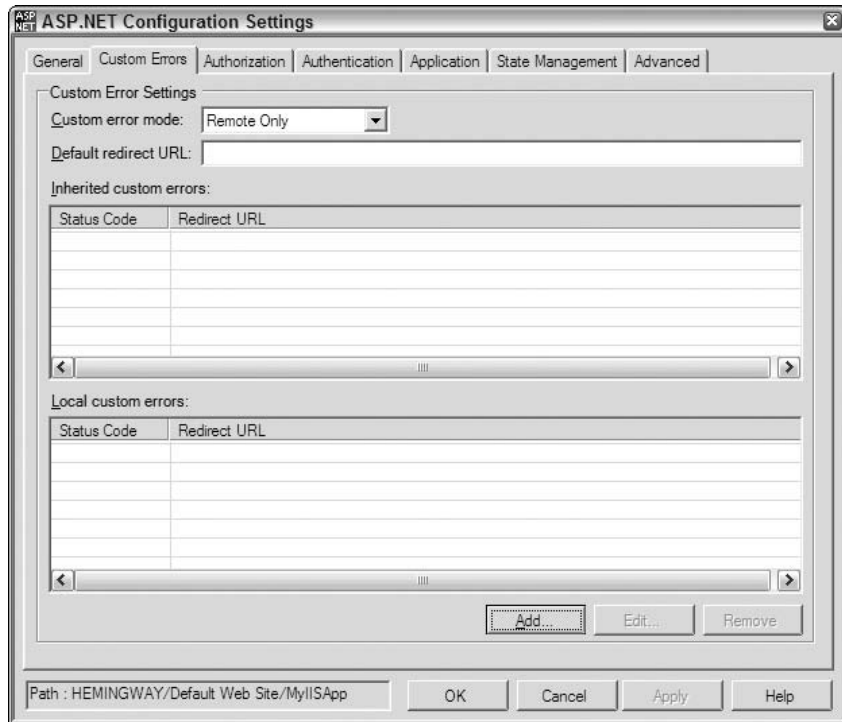


Figure 14-6

This particular tab allows you to work with the following items:

- ❑ **Custom error mode:** This drop-down list applies custom errors for particular users of the application. The default option is Remote Only. This option ensures that errors are redirected only for users who are on a remote machine. The other settings for this drop-down list include On and Off. On turns on the error redirection for all users, whereas the Off setting turns off the error redirecting for all users.
- ❑ **Default redirect URL:** The URL to which all errors are redirected.
- ❑ **Inherited custom errors:** All the errors that have been inherited from server defaults. These can be redirections for custom errors that are set in the `machine.config` file.
- ❑ **Local custom errors:** The errors that are set by you for this particular application. Error redirections are set using a name/value pair for Status Code/Redirect URL.

Authorization

The third tab is the Authorization tab. This section of the MMC enables you to authorize specific users or groups for the application (see Figure 14-7).

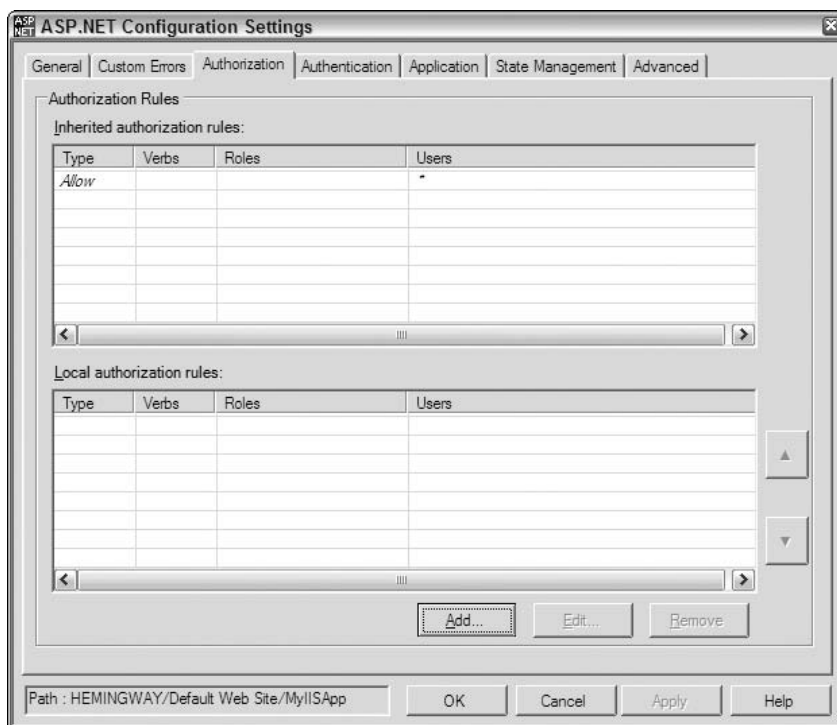


Figure 14-7

This section enables you to create roles made up of multiple users and/or groups. This dialog contains two items:

- ❑ **Inherited authorization rules:** All the authorization rules inherited from server defaults. These can be roles that are established in the `machine.config` file of the server.
- ❑ **Local authorization rules:** The authorization rules that you set for this particular application.

From this dialog, you can add, edit, or remove roles that have been created. If you click the Add button, the dialog shown in Figure 14-8 appears.

You can either allow or deny users access to the application by using the Edit Rule dialog. To use this feature, click the appropriate option in the Rule Type section.

The Verbs section allows you to apply a specific rule to those end users retrieving the page via all possible means (HTTP-POST or HTTP-GET), or to narrow the rule to cover only the specific verbs you want using the second option. Remember that the verb of a request is in how the request is actually made. The possible options specify that the request can be made either using HTTP-POST or HTTP-GET.

The final section, Users and Roles, enables you to choose who you want the rule to be applied to: all users that come to the site, anonymous users only, specific users, or users contained within specific groups.

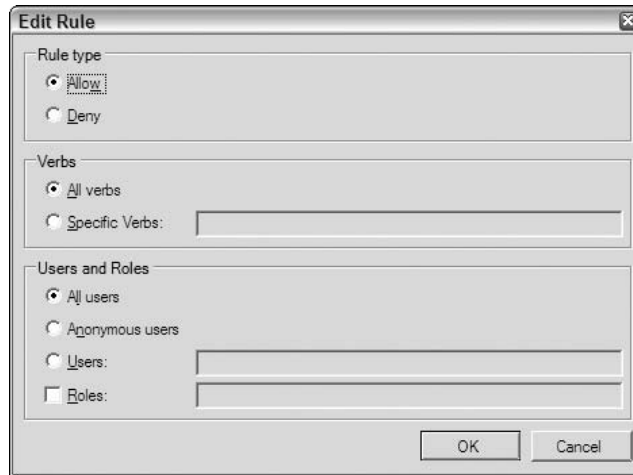


Figure 14-8

Authentication

The fourth tab in the MMC dialog is the Authentication tab (see Figure 14-9). This tab enables you to modify how your application will authenticate users for later authorization.

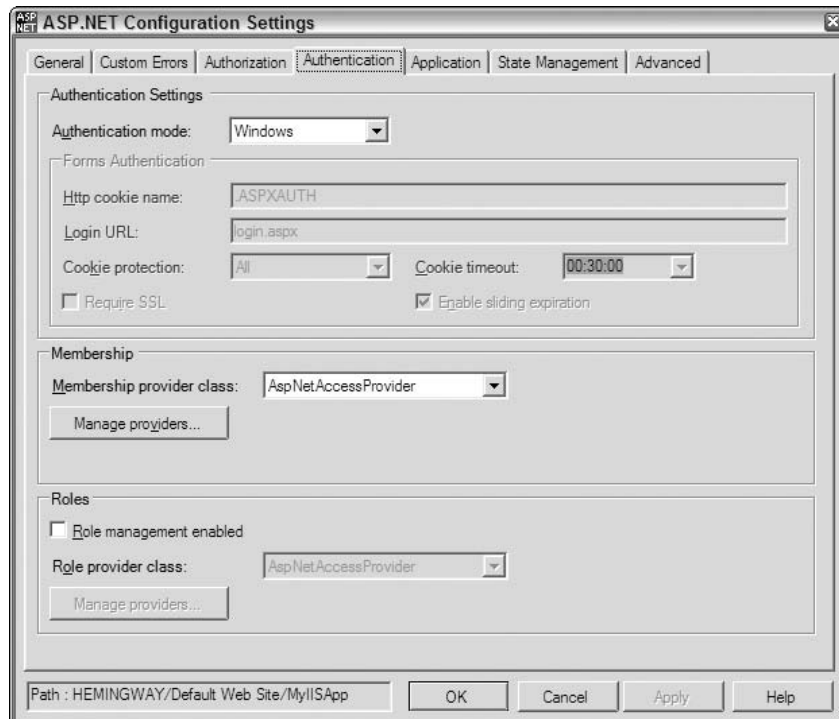


Figure 14-9

This dialog contains many options because you can work with the authorization of your end users in so many ways. The following list describes some of the items in this dialog:

- ❑ **Application Settings:** This section of the dialog lets you set the authentication mode of your application. The options in the drop-down list include Windows, Forms, Passport, or None. If you select Forms, the grayed-out options (shown in Figure 14-9) are available and enable you to modify all the settings that determine how forms authentication is applied.
- ❑ **Membership:** The Membership section enables you to tie the membership process to one of the available data providers available on your server. From this section, you can click the Manage Providers button to add, edit, or remove providers.
- ❑ **Roles:** From the Roles section, you can enable role-based management by checking the check box. From here, you can also tie the role management capabilities to a particular data provider.

Clicking the Manage Providers button opens the Provider Settings dialog (see Figure 14-10), which enables you to work with the data providers on the server.

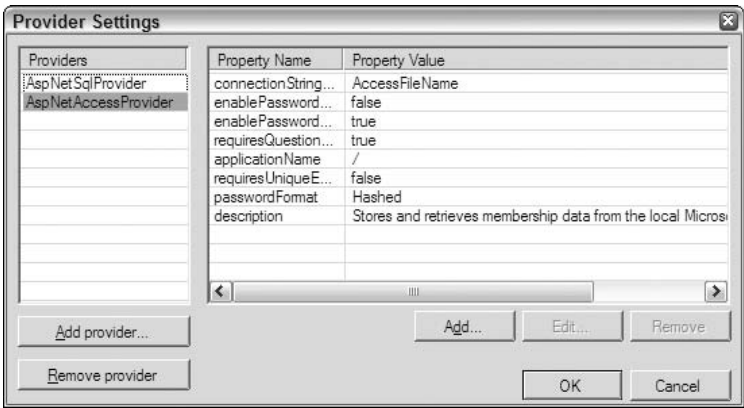


Figure 14-10

From the Provider Settings dialog, you can add, edit, or remove providers. You can also edit the settings of a particular provider. To edit any of the options in the dialog, just highlight the property that you want to change and click the Edit button. A new dialog pops up, which enables you to make changes.

Application

The fifth tab, Application, enables you to make more specific changes to the pages in the context of your application. From this dialog, shown in Figure 14-11, you can change how your pages are compiled and run. You can also make changes to global settings in your application.

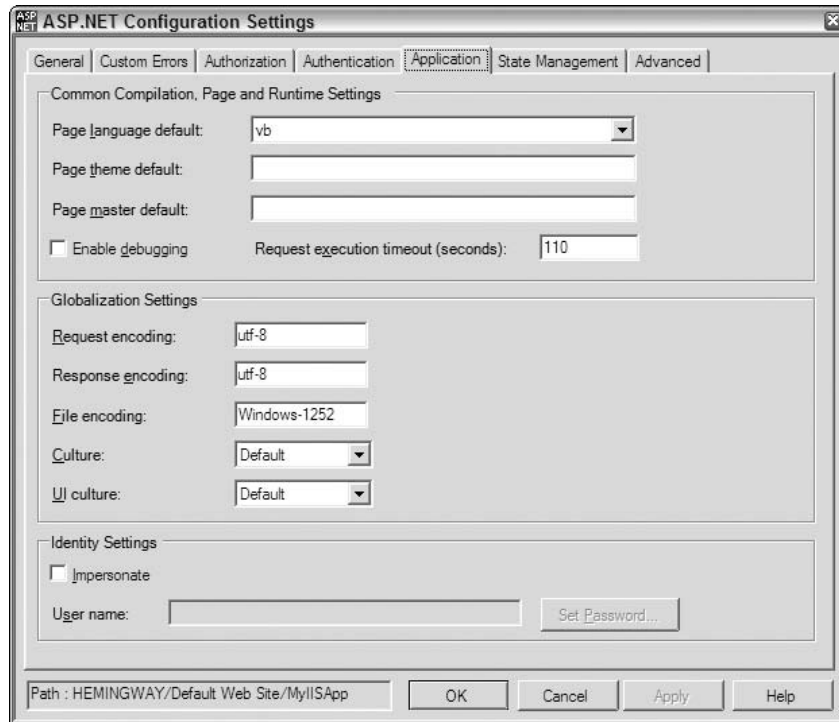


Figure 14-11

Again, this dialog provides you with a wealth of available options for modifying how the pages are run in a specific application as well as how your applications, in general, are built and run. The following list briefly describes some of these options:

- ❑ **Common Compilation, Page, and Runtime Settings:** This section includes a number of items that are very page-specific. From the first drop-down list, you can select the default language of your application. The available options include all the Microsoft .NET-compliant languages — C#, VB, JS, VJ#, and CPP. Other settings enable you to set the default theme or master page that your ASP.NET pages use during construction.
- ❑ **Globalization Settings:** This section allows you to set the default encodings and the cultures for your application.
- ❑ **Identity Settings:** The Identity Settings enable you to run the ASP.NET worker-process under a specific user account.

State Management

ASP.NET applications, being stateless in nature, are highly dependent on how state is stored. The sixth tab, the State Management tab (see Figure 14-12), enables you to change a number of different settings that determine how state management is administered.

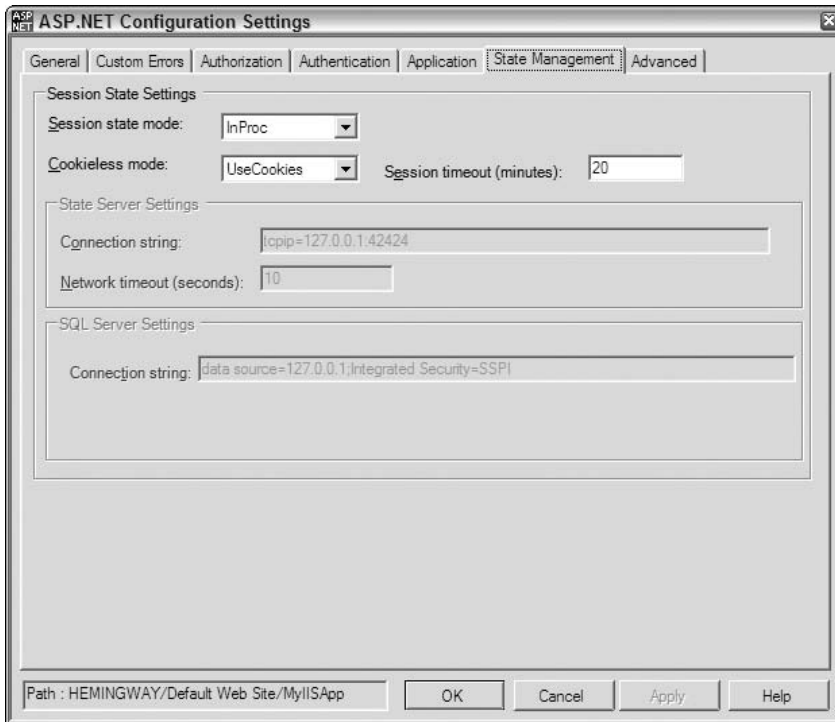


Figure 14-12

Because you can apply state management to your applications in a number of ways, this dialog allows for a number of different settings — some of which are enabled or disabled based on what is selected. The following list describes some of the available items in this dialog:

- ❑ **Session state mode:** This section enables you to determine how the sessions are stored by the ASP.NET application. The default option (shown in Figure 14-12) is InProc. Other options include Off, StateServer, and SQLServer. Running sessions in-process (InProc) means that the sessions are stored in the same process as the ASP.NET worker process. Therefore, if IIS is shut down and then brought up again, all the sessions are destroyed and unavailable to end users. StateServer means that sessions are stored out-of-process by a Windows service called ASPState. SQLServer is by far the most secure way to deal with your sessions@@it stores them directly in SQL Server itself. Although it is the most secure method, it is also the least performance-efficient method.
- ❑ **Cookieless mode:** The Cookieless mode section enables you to change how the identifiers for the end user are stored. The default setting uses cookies (UseCookies). Other possible settings include UseUri, AutoDetect, and UseDeviceProfile.
- ❑ **Session timeout:** Sessions are only stored for a short period of time before they expire. For years, the default has been 20 minutes. Changing the value here changes how long the sessions created by your application are valid.

Advanced

The last tab, the Advanced tab, is basically a catch-all for all other possible settings in the `web.config` file. The uppermost drop-down list in the dialog allows you to change the entire page of options. The available sections include

- ☐ Pages & Tracing
- ☐ Compilation
- ☐ Http Handlers
- ☐ Http Runtime
- ☐ Locations
- ☐ Http Modules
- ☐ Trust

I don't have room to discuss all of these sections in depth, but I cover some of the more interesting ones in the following paragraphs. Figure 14-13 shows the first option, Pages & Tracing, selected.

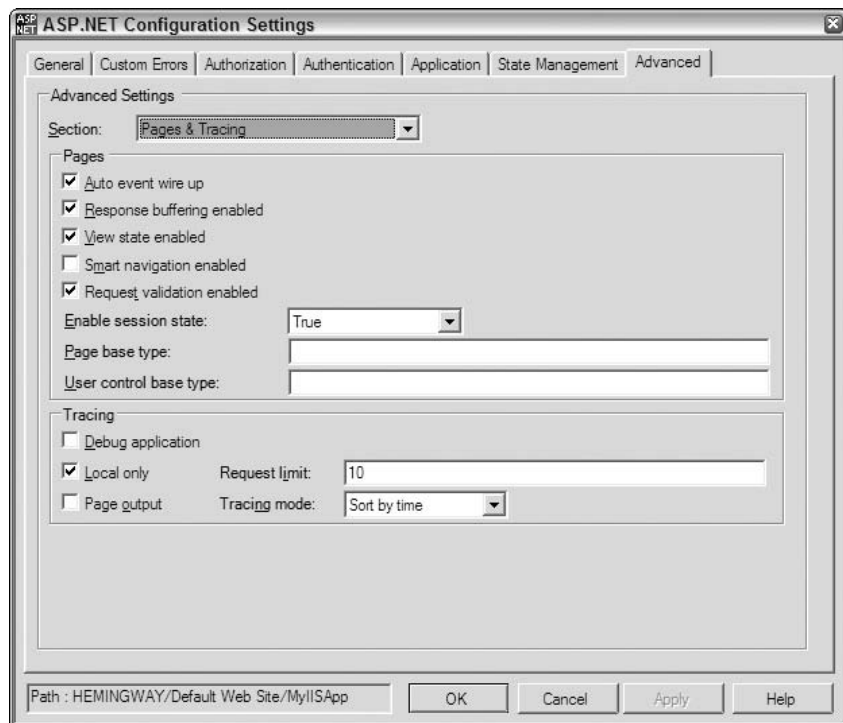


Figure 14-13

The Pages & Tracing page lets you determine how the pages are run and traced. You can decide whether smart navigation, view state, response buffering, or request validation is enabled. With tracing, you can set how tracing functions, including whether tracing is used for local requests only, and also how the tracing information is sorted.

The Compilation section of the Advanced tab enables you to define how your ASP.NET pages are compiled. For example, you can define whether Visual Basic’s Option Strict is always enabled or not enabled during the compilation process. You can also specify the local assemblies that are compiled with the application.

The Http Handlers section of the Advanced tab, shown in Figure 14-14, enables you to make modifications to the handlers available for your application. By default, a number of handlers — such as handlers for tracing and the ASP.NET Web Site Administration Tool — are already in place. In this section of the dialog, you can add, edit, or remove handlers.

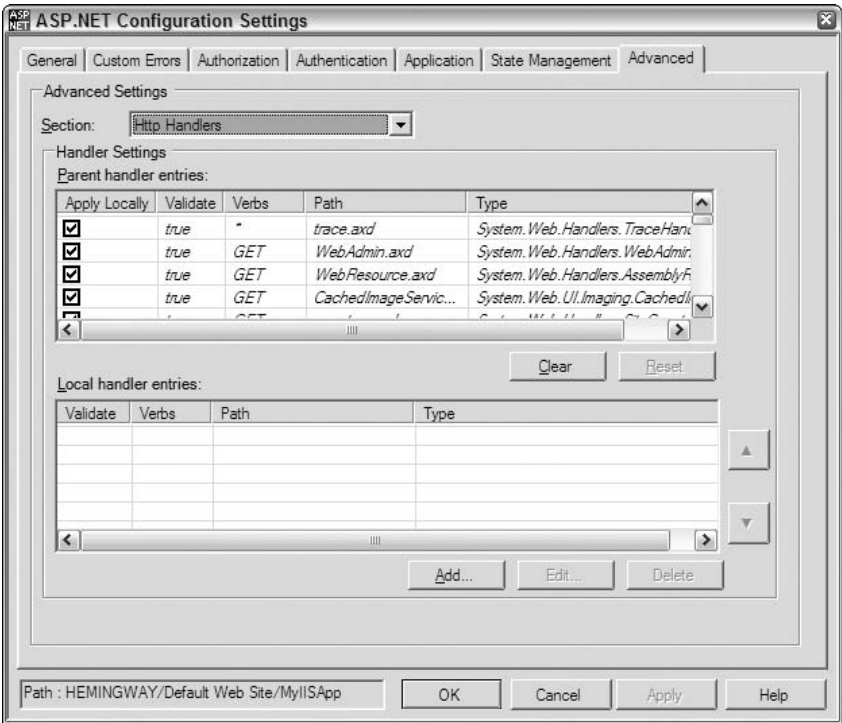


Figure 14-14

Another Advanced tab option, Http Runtime, is shown in Figure 14-15. This section enables you to set the maximum request length that your application can deal with. By default, this is set at around 4MB, but for security purposes it is always better to lower this number to something that are you still comfortable with. Other options in this dialog enable you to set the minimum number of free threads for new requests, the minimum number of free threads for new local requests, and the application request queue limit.

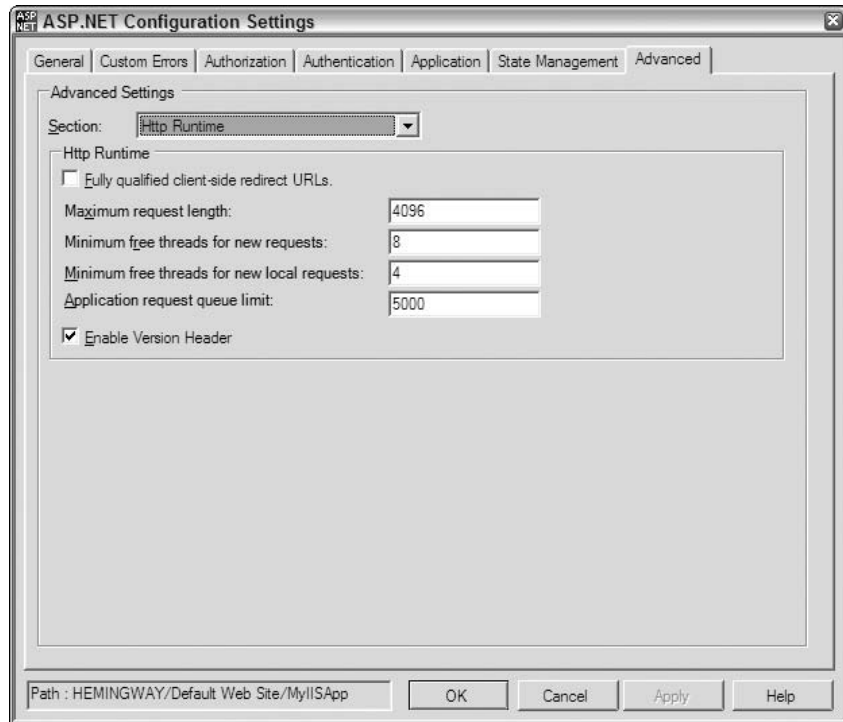


Figure 14-15

Although a number of other settings are possible through the MMC console, I will stop here. The point is that if you are an administrator of ASP.NET applications, gone are the days when you were required to go to XML files to fiddle with the settings. Fiddling is an error-prone method of administration and is effectively eliminated through the new administration GUIs — one of which is provided by the new ASP.NET MMC snap-in.

Next, I show you the other administration tool for ASP.NET — the ASP.NET Web Site Administration Tool.

ASP.NET Web Site Administration Tool

In addition to the new ASP.NET snap-in for the MMC console, another outstanding new GUI-based tool for administering your Web site is the ASP.NET Web Site Administration Tool (WAT). WAT lets you work in Visual Studio 2005 or directly from the browser to modify the settings stored within the application's `web.config` file.

By default, all local users can automatically use WAT to administer settings for their Web applications. The settings are primarily set and stored within your application's `web.config` file. If your application doesn't have a `web.config` file, WAT creates one for you. The changes that you make using WAT are immediately applied to the `web.config` file.

Chapter 14

You can get at WAT in a couple of ways. The first is to click the ASP.NET Configuration button in Visual Studio Solution Explorer. This opens up a new tab in the document window, which is basically just a browser view of WAT, as shown in Figure 14-16.

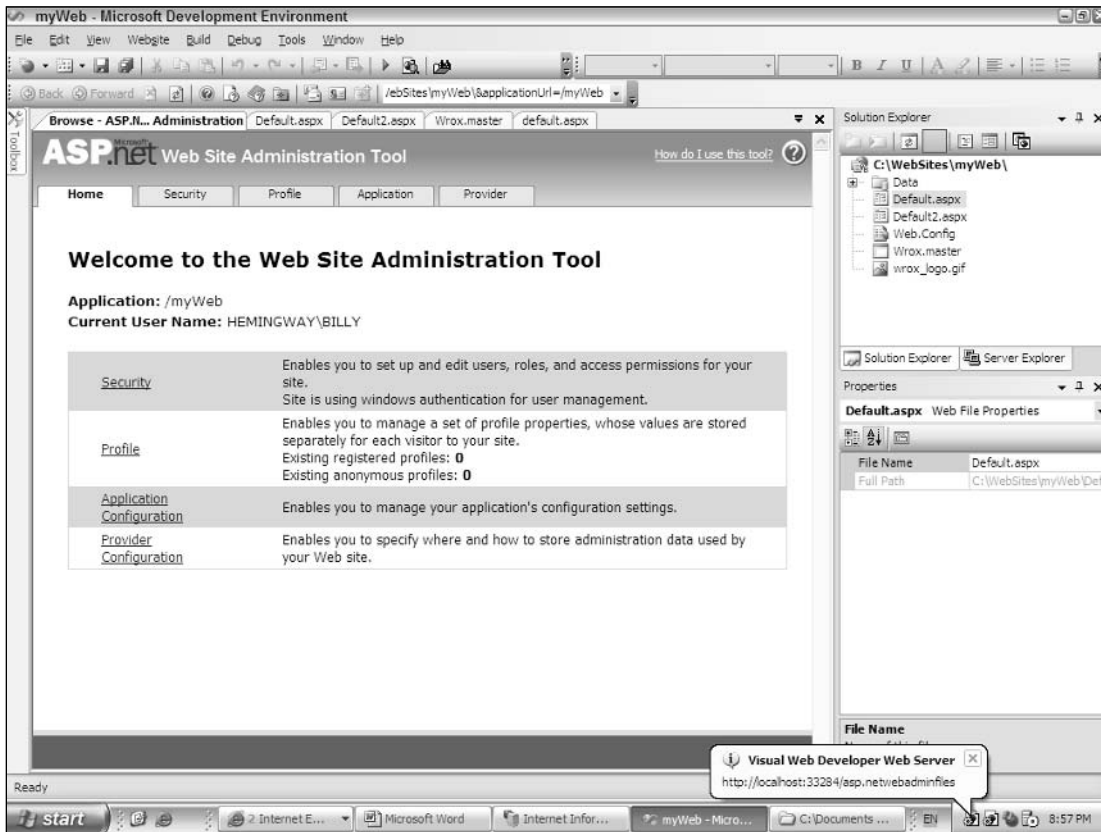


Figure 14-16

This figure shows the ASP.NET Web Site Administration Tool running in Visual Studio 2005 as one of the tabs in the document window. You can see from the screen shot that ASP.NET also fires up a new instance of the built-in ASP.NET Web Server to run the tool. Using WAT directly in Visual Studio allows you to change your application's settings in a GUI-fashion, as opposed to working from an XML file (as you did with ASP.NET 1.0/1.1).

Another option for working with WAT is to pull it up directly in the browser instead of working with it in Visual Studio. To do this, you must call the appropriate http handler. The following URL shows what you type into the browser for an application called `myWeb`:

```
http://localhost/myWeb/Webadmin.axd
```

Typing this line gives you the results shown in Figure 14-17.

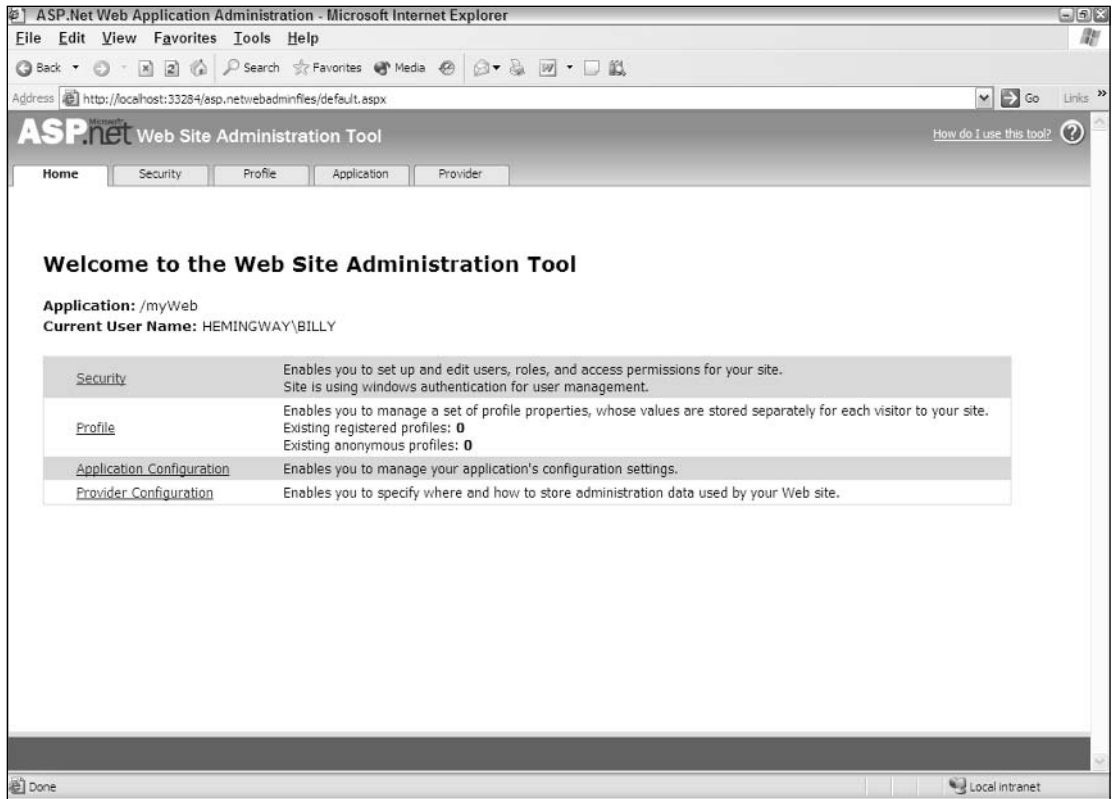


Figure 14-17

You can also get at WAT by clicking Website ⇄ ASP.NET Configuration in the menu bar of Visual Studio. Whether you pull up WAT in Visual Studio or directly in the browser makes no difference. Accessing WAT directly from Internet Explorer is ideal for developers who are working with remote or hosted applications and must change settings on the fly.

Now that you know how to get at WAT, take a look at each of the sections that it provides.

Home

WAT is made up of five tabs. The first tab, Home, is a summary tab that provides you with some basic information about the application you are monitoring or modifying.

Using this tab, you can see the name of the application and the current user context in which you are accessing the application. In addition to these basic items, you see links to the other four tabs of the WAT application — some of which provide you with a summary of the settings contained in them. To make changes to your application, you click the appropriate tab or link.

Security

The second tab, Security (see Figure 14-18), enables you to set up the authentication aspect of your ASP.NET application. You can set the authentication type, roles, and rules for your application.

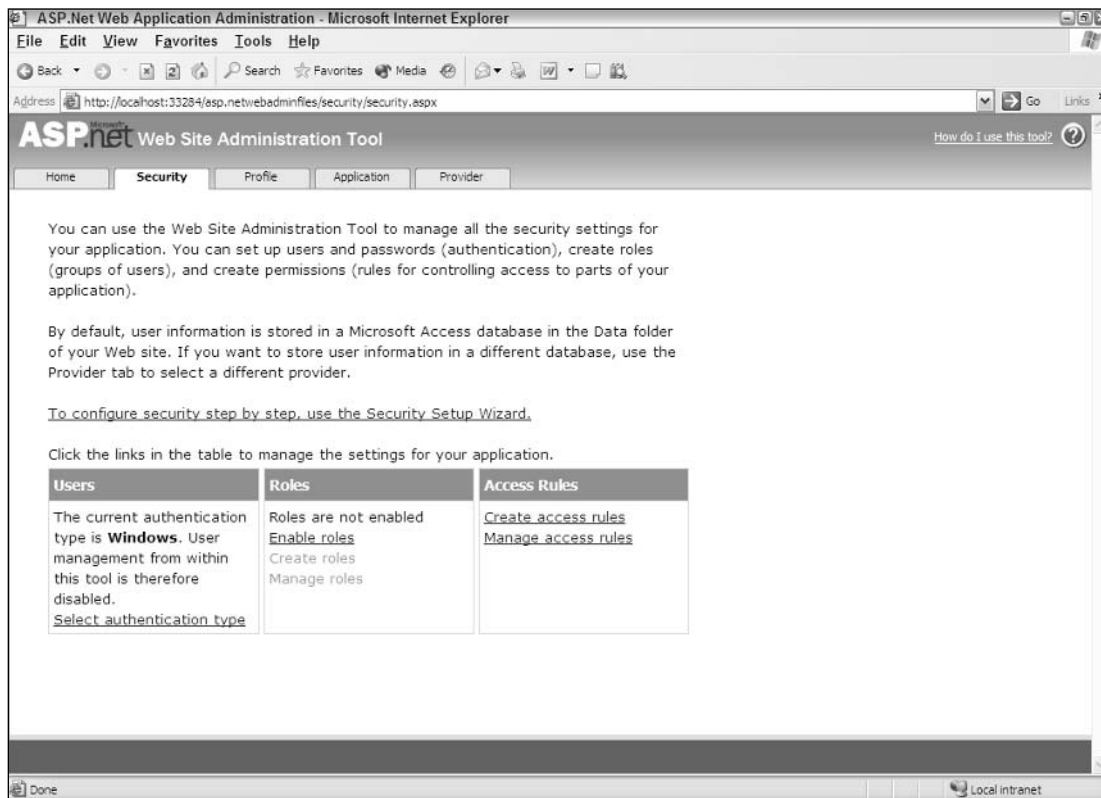


Figure 14-18

At the bottom of the page on the Security tab, the Select Authentication Type link brings you to a new page that asks whether your application is going to run on a network or be open to the public. If you choose to run your application on a closed network, Windows authentication is applied. If you say that your application will be exposed to the Internet, your application is configured for Forms authentication.

The Roles section enables to create roles that you can apply to your application. First you have to click the Enable Roles link that enables the Create Roles and Manage Roles links. With these two links, you can create roles (by giving a role a specific name) and then assign users to the roles you create.

The Access Rules section offers an outstanding way to give or deny access to certain roles or users down to the folder level. After you create a rule, you can edit or delete it.

The other option on the Security tab is the Security Setup Wizard. This wizard, shown in Figure 14-19, makes it even easier to set up the users, roles, and rules because it takes you through a step-by-step process.

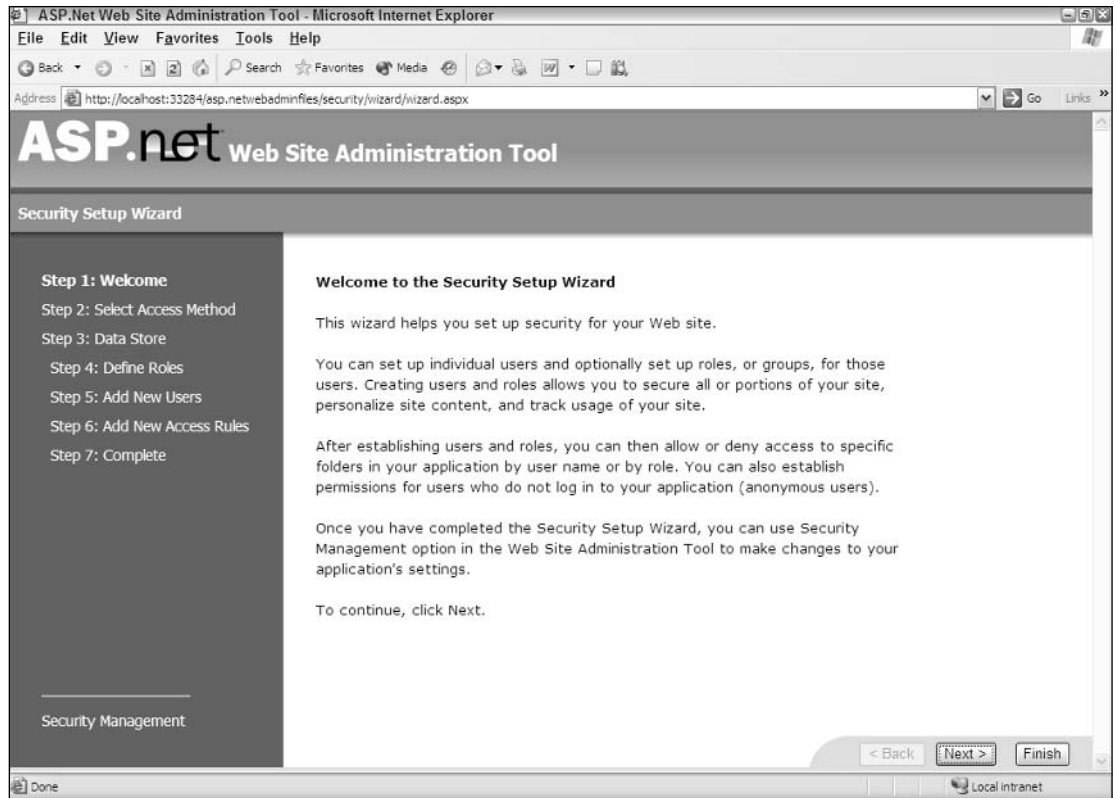


Figure 14-19

From the first page of the Security Setup Wizard, you see steps listed for a number of different tasks. These steps enable you to select the Windows or Forms authentication, choose the data provider you want to use, define the roles, add users, and then create rules for your application.

Profile

Using the Profile tab, shown in Figure 14-20, you can set up your application to work with the new personalization management system. The personalization system allows you to store unique values for authenticated or unauthenticated users.

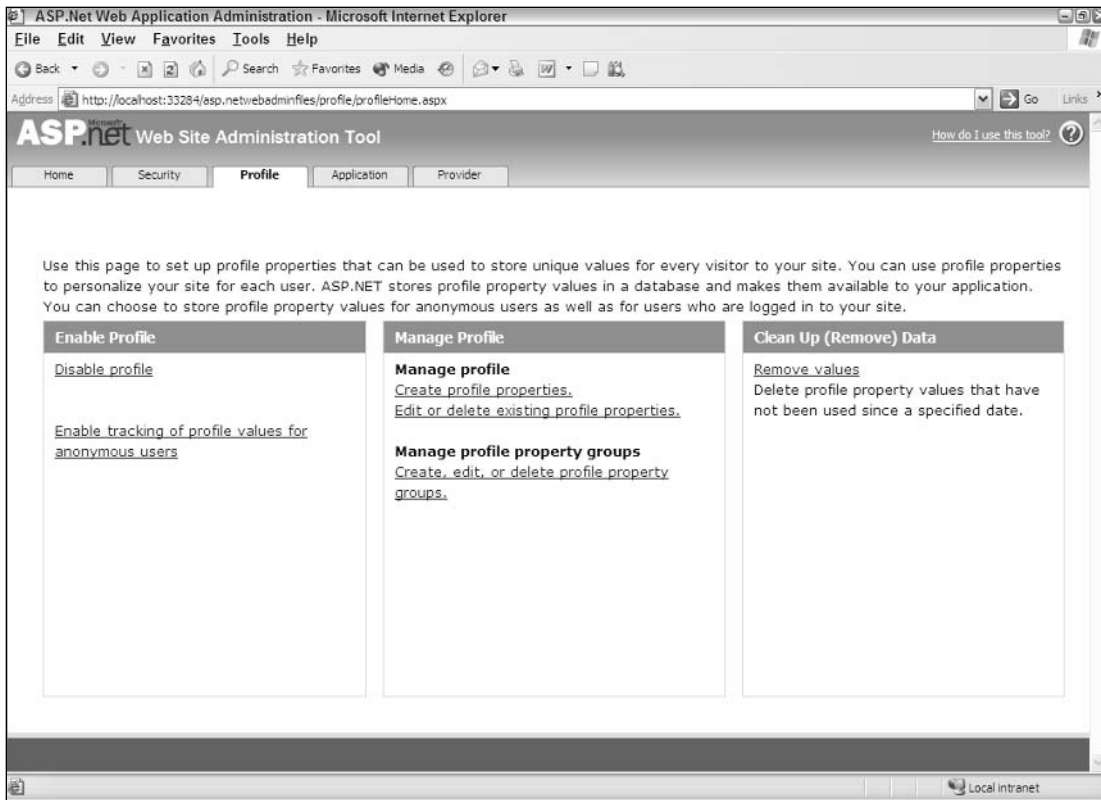


Figure 14-20

The ASP.NET personalization features are enabled by default; therefore, the Enable Profile section lets you disable personalization if you want. It also allows you to enable personalization for anonymous users — something that is disabled by default. With the Manage Profile section, you create, edit, or delete profile properties or groups of properties. The last section, Clean Up (Remove) Data, enables you to set a date when personalization points become stale. This setting removes the personalization points from your data store if they are older than the date specified.

Application

The Application tab, shown in Figure 14-21, enables you to create application settings (key/value pairs), configure site counters, apply settings so your application can send e-mails, as well as modify debugging, tracing, and error page settings.

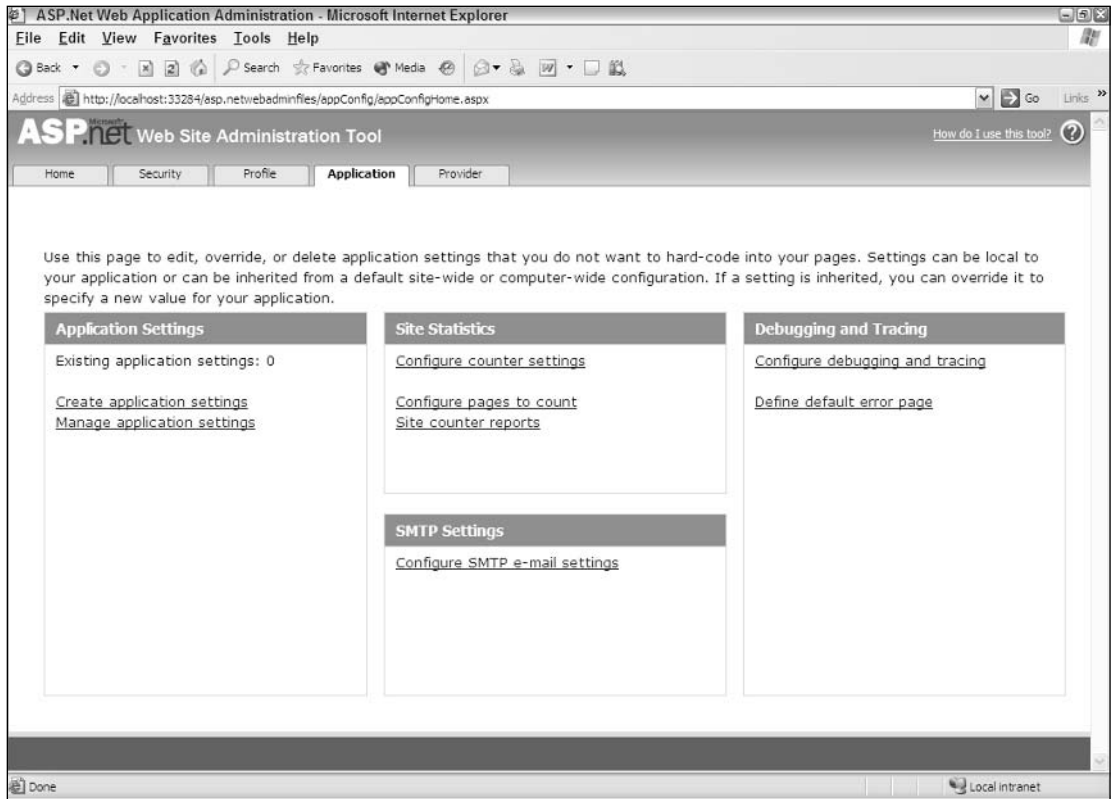


Figure 14-21

Provider

The final tab in the Web Admin tool is the Provider tab (see Figure 14-22). You use this tab to set up additional providers and to determine the providers your application will use.

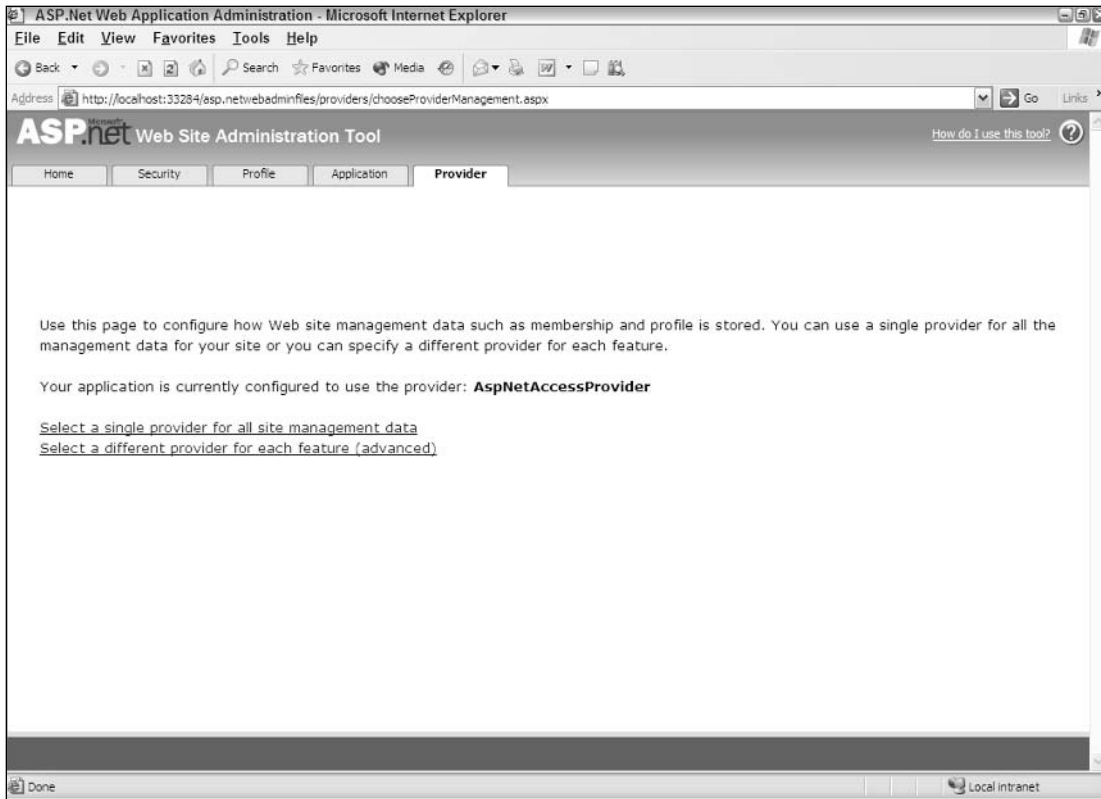


Figure 14-22

This Provider page is simple, but it contains one important piece of information: the default data provider your application is geared to work with. Figure 14-22 shows that my application is set up to work with the `AspNetAccessProvider`, the default data provider.

The two links on this tab let you set up either a single data provider (see Figure 14-23) or a specific data provider for each of the features in ASP.NET that requires a data provider.

As you can see from the screen shots and brief explanations provided here, you can now handle a large portion of the necessary configurations through a GUI. You no longer have to figure out which setting must be placed in the `web.config` file. This functionality becomes even more important as the `web.config` file grows. In ASP.NET 1.0/1.1, the `web.config` file possibilities were a reasonable size; but with all the new features now provided by ASP.NET 2.0 the `web.config` file has the potential to become very large. The new GUI-based tools are an outstanding way to manage it.

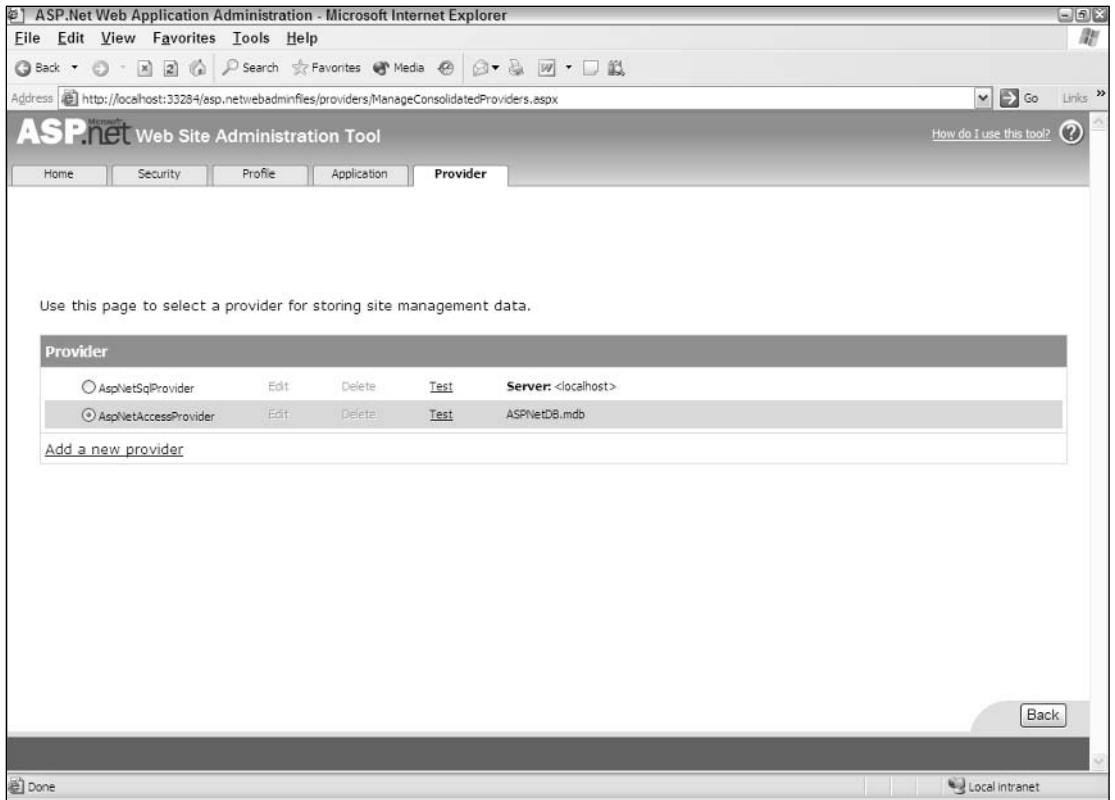


Figure 14-23

Managing the Site Counter System

Another outstanding management feature is ASP.NET's capability to monitor the views and clicks occurring in an application. Not only does the new site counter system count the views and clicks that occur as end users work through your Web application, it also generates reports in the browser with the site counter results.

To see an example of the new site counter system, begin by working with the ASP.NET Web Site Administration Tool. Open WAT and click the Configure Counter Settings link on the Application tab. You see a page dealing with counters, as shown in Figure 14-24.

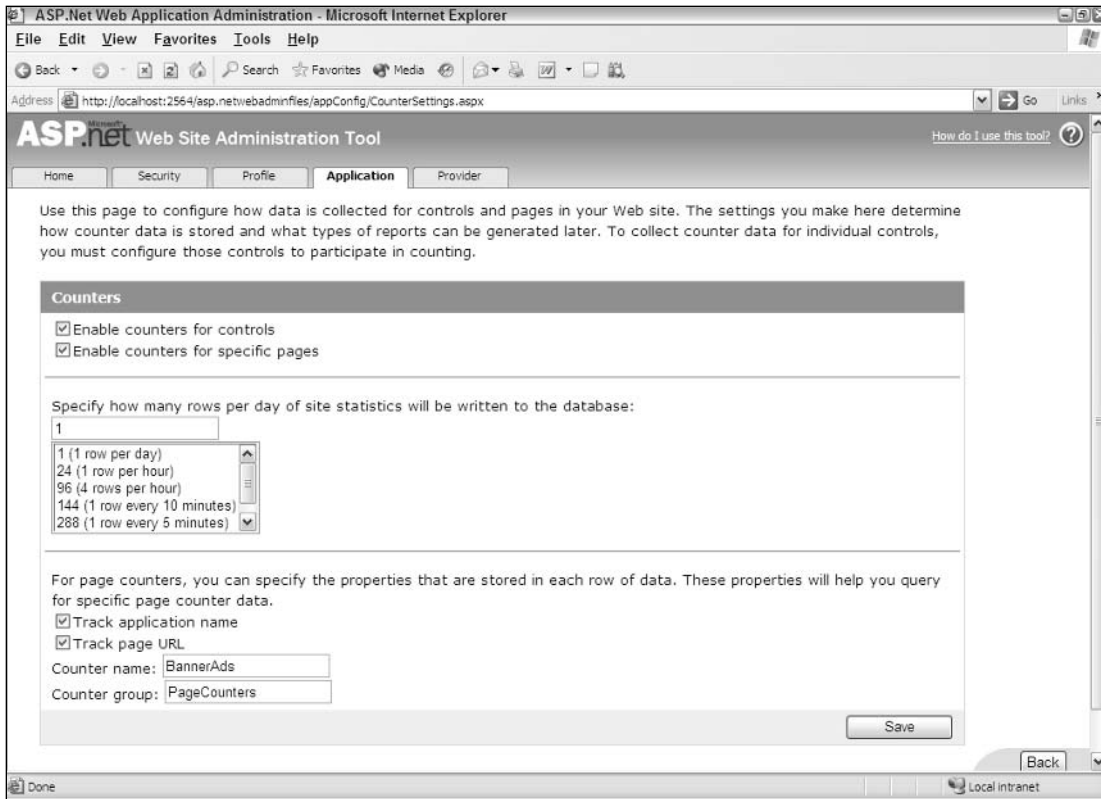


Figure 14-24

On this screen, make sure that the Enable Counter For Controls check box is checked. Then, in the text box labeled Specify How Many Rows Per Day of Site Statistics Will Be Written to the Database, keep the default of 1. One row in the database is used to store the counts. For the rest of the options, make sure the Track Application Name and Track Page URL check boxes are checked. Give your created site counter configuration a name and associate it to a counter group. As you can see in Figure 14-24, I gave the counter the name `BannerAds` and the counter group the name `PageCounters`. Finally, save your configuration by clicking the Save button in the lower-right corner of the dialog.

This operation has changed your `web.config` file. The results are shown in Listing 14-1.

Listing 14-1: Enabling the web.config for the site counter system

```
<?xml version="1.0"?>
<configuration>
  <system.web>

    <siteCounters defaultProvider="AspNetAccessProvider" enabled="true"
      rowsPerDay="1">
      <pageCounters enabled="true" trackApplicationName="true"
        trackPageUrl="true" counterName="BannerAds">
```

```

        counterGroup="PageCounters" />
    </siteCounters>

</system.web>
</configuration>

```

Remember that although it is possible to work with tools such as WAT to build all this for you in the application's configuration file, you can also place all the information in the `web.config` file yourself. The best approach is whatever is easier for you in the end.

Now that you have configured the `web.config` file to work with control clicks, learn how to monitor button clicks and views of an `AdRotator` server control. This is a common scenario because many Web sites sell advertising and charge their clients based on views, clicks, or both. Not only will the site counter system tally these items for you, but it also provides the GUI reports of what was tallied. Listing 14-2 shows an `AdRotator` control enabled to work with the site counter system.

Listing 14-2: A server control working with the site counter system

```

<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>AdRotator Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:AdRotator ID="AdRotator1" Runat="server" AdvertisementFile="MyAds.xml"
            CountViews="true" CountClicks="true" />
        <p>Lorem ipsum dolor sit
            amet, consectetur adipiscing elit. Duis vel justo. Aliquam
            adipiscing. In mattis volutpat urna. Donec adipiscing, nisl eget
            dictum egestas, felis nulla ornare ligula, ut bibendum pede augue
            eu augue. Sed vel risus nec urna pharetra imperdiet. Aenean
            semper. Sed ullamcorper auctor sapien. Suspendisse luctus. Ut ac
            nibh. Nam lorem. Aliquam dictum aliquam purus.</p>
    </form>
</body>
</html>

```

This `AdRotator` server control isn't much different from the `AdRotator` server control shown in Listing 13-7 in the last chapter — although this one has a couple of new attributes. To work with the site counter system provided by ASP.NET 2.0, you add the `CountViews` and the `CountClicks` attributes. In both cases, you set these attribute values to `True`. The `CountViews` attribute enables or disables the site counter system, which records each and every time that the image shown by the `AdRotator` control is actually viewed in the browser. The `CountClicks` attribute is always a lesser number because it enables or disables the site counter system that counts the number of times an end user clicks on the advertisement.

Now that both of these attributes have been set to `True` and the `web.config` file is configured properly, you can run the application and refresh the page a few times. Click the advertisement generated by the `AdRotator` control a few times to generate some results for the site counter system.

Chapter 14

Then, with the ASP.NET Web Site Administration Tool, return to the Application tab and click the Site Counter Reports link. This pulls up details on the PageCounter's Details page. From this page, you can see the number of page views that have been counted by the system. From the drop-down list on the page, select AdRotator. This causes WAT to pull up details about the AdRotator server control, as shown in Figure 14-25.

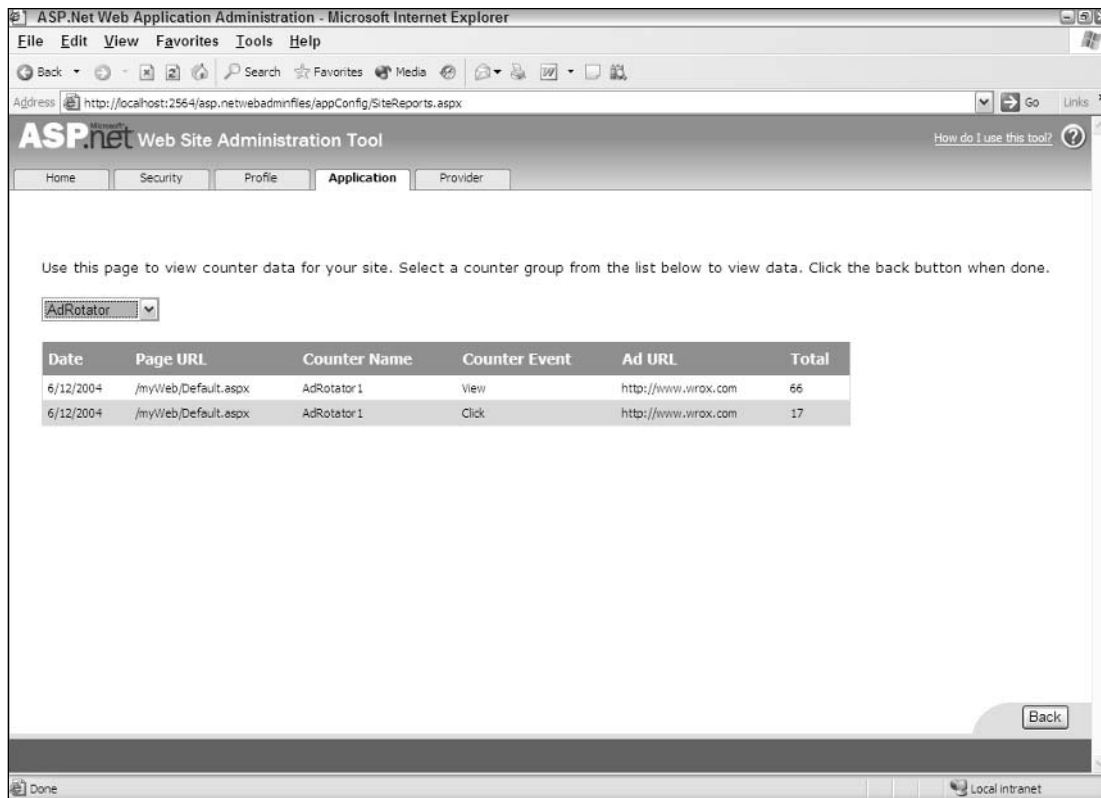


Figure 14-25

On this page, you can see the date the views or clicks were recorded, the page the details came from, and the ID of the server control (in this case AdRotator1). It also shows the count event (Views or Clicks), the destination URL of the control, and finally the number recorded by the site counter system.

Summary

This chapter showed you some of the new management tools that come with the latest release of ASP.NET. These new tools make the ever-increasing size of the `web.config` file more manageable because the tools take care of setting the appropriate values in the application's configuration file.

The ASP.NET snap-in to the Microsoft Management Console is a welcome addition for managing applications that are configured to work with IIS. The ASP.NET Web Site Administration Tool provides even more value to administrators and developers as it enables them to remotely manage settings.

Finally, you learned how to use the site counter system to create detailed reports on the clicks and views that your Web application produces.

15

Visual Basic 8.0 and C# 2.0 Language Enhancements

A lot has changed with the *Whidbey* release of .NET. Not only are there dramatic changes to ASP.NET (as I have shown you throughout this book), but considerable changes have been made to the IDE, Windows Forms, Visual Basic, C#, and more. This chapter focuses on the changes to Visual Basic and C# languages because these are the two languages most commonly used for ASP.NET development. Because of their heavy use in Web application development, it is vital to understand the capabilities of these languages and the direction they are taking.

Probably one of the greatest changes to Web application development in the Microsoft world is .NET's use of true object-oriented languages such as Visual Basic .NET and C# to build Web applications. You are no longer required to work with interpreted languages. Although they have only recently been introduced to the Web application world, these object-oriented languages are continuing to evolve, bringing new features to Web application development.

This last chapter focuses on the changes that have occurred to both Visual Basic and C# with this latest release of the .NET Framework. You can apply what you learn here directly to your ASP.NET 2.0 applications.

Overview of Changes

Both Visual Basic and C# have undergone changes with the release of the .NET Framework 2.0. Some of the changes have occurred in both languages, whereas other changes have occurred in only one.

Throughout the book I refer to the VB language as Visual Basic. With this release of the .NET Framework, the language has reverted to the name Visual Basic (minus the .NET at the end of the name). This version of the VB language is called Visual Basic 8.0, whereas the newest version of C# is 2.0.

Some new features of these two languages include those described in the following table.

New Language Feature	Visual Basic 8.0	C# 2.0
Generics	Yes	Yes
Iterators	No	Yes
Anonymous methods	No	Yes
Operator overloading	Yes	Yes (already available)
Partial classes	Yes	Yes
XML documentation	Yes	Yes (already available)

Take a look at some of these new features and how to use them in your applications.

Generics

In order to make collections a more powerful feature and also increase their efficiency and usability, generics were introduced to both Visual Basic and C#. The idea of *generics* is nothing new. They are similar to C++ templates. You can also find generics in other languages, such as Java. Their introduction into the .NET Framework 2.0 languages is a huge benefit for the user.

Generics enable you to make a generic collection that is still strongly typed — providing fewer chances for errors (because they occur at runtime), increasing performance, and giving you IntelliSense features when you are working with the collections.

First, look at the problems that can arise in a collection that does not use generics. Listing 15-1 shows a simple use of the `Stack` and `Array` classes.

Listing 15-1: A collection that doesn't use generics

```
VB
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myStack As New Stack
    myStack.Push("St. Louis Rams")
    myStack.Push(5)

    Dim myArray As Array
    myArray = myStack.ToArray()

    For Each item As String In myArray
        Label1.Text += item & "<br />"
    Next
End Sub
```

C#

```
void Page_Load(object sender, EventArgs e)
{
    Stack myStack = new Stack();
    myStack.Push("St. Louis Rams");
    myStack.Push(5);

    Array myArray;
    myArray = myStack.ToArray();

    foreach (string item in myArray)
    {
        Label1.Text += item + "<br />";
    }
}
```

In this code example, you can see two distinct items in the `Stack`: a `String` with the value of `St. Louis Rams` and an `Integer` with a value of 5. The `Stack` itself is not the best performing item in the world simply because it is an object-based list of items — meaning that anything (as you can see in the preceding listing) can be placed in the list of items. When the `For Each` section is reached, however, the items are cast to a `String` value and displayed in a `Label` control. The Visual Basic example actually takes the 5, which should be an `Integer`, and casts it to a `String` and displays the `St. Louis Rams` and the 5 as a `String` in the browser. The C# example does not cast the 5 as a `String`, but instead throws an exception on the cast at runtime.

Generics enable you to create type-specific collections. The `System.Collections.Generic` namespace gives you access to generic versions of the `Stack`, `Dictionary`, `SortedDictionary`, `List`, and `Queue` classes. Again, you can make these collections type-specific to produce collections that perform better and that have design-time error checks and better IntelliSense features.

Listing 15-2 shows you how to create a generic version of the `Stack` class that includes a collection of `Strings`.

Listing 15-2: A generic Stack class

VB

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myStack As New Generic.Stack(Of String)
    myStack.Push("St. Louis Rams")
    myStack.Push("Indianapolis Colts")
    myStack.Push("Minnesota Vikings")

    Dim myArray As Array
    myArray = myStack.ToArray()

    For Each item As String In myArray
        Label1.Text += item & "<br />"
    Next
End Sub
```

(continued)

Listing 15-2: (continued)

C#

```
void Page_Load(object sender, EventArgs e)
{
    System.Collections.Generic.Stack<string> myStack =
        new System.Collections.Generic.Stack<string>();
    myStack.Push("St. Louis Rams");
    myStack.Push("Indianapolis Colts");
    myStack.Push("Minnesota Vikings");

    Array myArray;
    myArray = myStack.ToArray();

    foreach (string item in myArray)
    {
        Label1.Text += item + "<br />";
    }
}
```

In the example in Listing 15-2, the `Stack` class is explicitly cast to be a collection of type `String`. In Visual Basic, you do this by following the collection class with `(Of String)` or `(Of Integer)` or whatever type you want to use for your collection. In C#, you specify the collection type with the use of brackets. You cast the `Stack` class to type `string` using `Stack<string>`. If you want to cast it to a `Stack` collection of type `int`, you specify `Stack<int>`.

Because the collection of items in the `Stack` class is cast to a specific type immediately as the `Stack` class is created, the `Stack` class no longer casts everything to type `Object` and then later (in the `For Each` loop) to type `String`. This process is called *boxing*, and it is expensive. Because you specify the types upfront, you increase performance for your collections.

Remember that when working with generic collections (as shown in the previous code example), you must import the `System.Collections.Generic` namespace into your ASP.NET page.

Now, change the `Stack` class from Listing 15-2 so that instead of working with `String` objects, it uses `Integer` objects in the collection. This change is illustrated in Listing 15-3.

Listing 15-3: A generic Stack class using Integers

VB

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myStack As New Generic.Stack(Of Integer)
    myStack.Push(5)
    myStack.Push(3)
    myStack.Push(10)

    Dim myArray As Array
    myArray = myStack.ToArray()

    Dim x As Integer = 0
    For Each item As Integer In myArray
        x += item
    
```

```
Next

    Label1.Text = x.ToString()
End Sub
```

C#

```
void Page_Load(object sender, EventArgs e)
{
    System.Collections.Generic.Stack<int> myStack =
        new System.Collections.Generic.Stack<int>();
    myStack.Push(5);
    myStack.Push(3);
    myStack.Push(10);

    Array myArray;
    myArray = myStack.ToArray();

    int x = 0;
    foreach (int item in myArray)
    {
        x += item;
    }

    Label1.Text = x.ToString();
}
```

The `Stack` class used in Listing 15-3 specifies that everything contained in its collection must be of type `Integer`. In this example, the numbers are added together and displayed in the `Label` control.

You can also use generics with classes, delegates, methods, and more. This is also an exciting way to apply generics. For an example, you can create a method that utilizes generics and, therefore, can work with any type thrown at it. The use of generics in methods is illustrated in Listing 15-4.

Listing 15-4: A generic method

VB

```
Public Function GenericReturn(Of ItemType)(ByVal item As ItemType) As ItemType
    Return item
End Function
```

C#

```
public ItemType GenericReturn<ItemType>(ItemType item)
{
    return item;
}
```

This simple method returns the value that is passed to it. The value can be of any type. To construct a generic method, you must follow the method name with `(Of ItemType)` in Visual Basic or `<ItemType>` in C#. This specifies that the method is indeed a generic method.

Chapter 15

The single parameter passed into the method is also of `ItemType` and the return value is the same as the type that is established when the method is called. In Listing 15-5, note how you go about calling this generic method.

Listing 15-5: Invoking the generic method

VB

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = GenericReturn(Of String)("Hello there!")
    Label2.Text = (GenericReturn(Of Integer)(5) + 5).ToString()
End Sub
```

C#

```
void Page_Load(object sender, EventArgs e)
{
    Label1.Text = GenericReturn<string>("Hello there!");
    Label2.Text = (GenericReturn<int>(5) + 5).ToString();
}
```

This little example in Listing 15-5 shows two separate invocations of the `GenericReturn` method. The first instance populates the `Label1` control and invokes the `GenericReturn` method as a `String`, which is quickly followed by the `String` value that is passed in as the `item` parameter. When called in this manner, the method is invoked as if it were constructed as

```
Public Function GenericReturn(ByVal item As String) As String
    Return item
End Function
```

or

```
public string GenericReturn(string item)
{
    return item;
}
```

The second invocation of the `GenericReturn` method passes in an object of type `Integer`, adds 5, and then uses that value to populate the `Label2` control. When called in this manner, the method is invoked as if it were constructed as

```
Public Function GenericReturn(ByVal item As Integer) As Integer
    Return item
End Function
```

or

```
public int GenericReturn(int item)
{
    return item;
}
```

As you can see, you gain a lot of power using generics. You see generics used in both of the main .NET languages because they can be built into the underlying framework.

Iterators

Iterators enable you to specify how your classes or collections work when they are dissected in a `foreach` loop. The iterators are used only in C#. Visual Basic 8.0 developers do not have a similar feature at present.

You can iterate through a collection of items just as you have always been able to do in C# 1.0 because the item implements the `GetEnumerator` function. For example, you can just run a `foreach` loop over an `ArrayList`, as shown in Listing 15-6.

Listing 15-6: Running the `foreach` loop over an `ArrayList`

```
void Page_Load(object sender, EventArgs e)
{
    ArrayList myList = new ArrayList();

    myList.Add("St. Louis Rams");
    myList.Add("Indianapolis Colts");
    myList.Add("Minnesota Vikings");

    foreach (string item in myList)
    {
        Response.Write(item.ToString() + "<br />");
    }
}
```

This code writes all three values that were added to the `ArrayList` to the browser screen. Iterators enable you to run a `foreach` loop on your own items such as classes. To run a `foreach` loop, you create a class that implements the `IEnumerable` interface.

The first step is to create a class in your Web solution. To create a class, create a folder in your solution and give it the name `Code`. Then place a new `.cs` class file in the `Code` directory. This class is illustrated in Listing 15-7.

Listing 15-7: Creating a class that works with a `foreach` loop

```
using System;
using System.Collections;

public class myList
{
    internal object[] elements;
    internal int count;

    public IEnumerator GetEnumerator()
    {
        yield return "St. Louis Rams";
        yield return "Indianapolis Colts";
        yield return "Minnesota Vikings";
    }
}
```

Chapter 15

This class, `myList`, imports the `System.Collections` namespace so that it can work with the `IEnumerable` interface. In its simplest form, the `myList` class implements the enumerator pattern with a method called `GetEnumerator()`, which returns a value defined as `IEnumerator`. Then each item in the collection is returned with the `yield return` command. The `yield` keyword in C# is used to provide a value to the enumerator object or to signal the end of the iteration.

Now that the class `myList` is in place, you can then instantiate the class and iterate through the class collection using the `foreach` loop. This is illustrated in Listing 15-8.

Listing 15-8: Iterating though the `myList` class

```
void Page_Load(object sender, EventArgs e)
{
    myList IteratorList = new myList();

    foreach (string item in IteratorList)
    {
        Response.Write(item.ToString() + "<br />");
    }
}
```

This ASP.NET `Page_Load` event simply creates an instance of the `myList` collection and iterates through the collection using a `foreach` loop. This is all possible because an `IEnumerator` interface was implemented in the `myList` class. When you run this page, each of the items returned from the `myList` class using the `yield return` command displays in the browser.

One interesting change you can make in the custom `myList` class is to use the new generics capabilities provided by C#. Because you know that only `string` types are being returned from the `myList` collection, you can define that type immediately to avoid the boxing and unboxing that occurs using the present construction. Listing 15-9 shows the changes you can make to the class that was first presented in Listing 15-7.

Listing 15-9: Creating a class that works with a `foreach` loop using generics

```
using System;
using System.Collections;
using System.Collections.Generic;

public class myList : IEnumerable<string>
{
    internal object[] elements;
    internal int count;

    public IEnumerator<string> GetEnumerator()
    {
        yield return "St. Louis Rams";
        yield return "Indianapolis Colts";
        yield return "Minnesota Vikings";
    }
}
```

Anonymous Methods

Anonymous methods enable you to put programming steps within a delegate that you can later execute instead of creating an entirely new method. This can be handled in a couple different ways. You should note that anonymous methods are only available in C# and are not present in Visual Basic 8.0.

Without using anonymous methods, create a delegate that is referencing a method found elsewhere in the class file. In the example from Listing 15-10, when the delegate is referenced (by a button-click event), the delegate invokes the method that it points to.

Listing 15-10: Using delegates in a traditional manner

```
public partial class Default_aspx
{
    void Page_Load(object sender, EventArgs e)
    {
        this.Button1.Click += ButtonWork;
    }

    void ButtonWork(object sender, EventArgs e)
    {
        Label1.Text = "Welcome to the camp, I guess you all know why you're here.";
    }
}
```

In the example in Listing 15-10, you see a method in place called `ButtonWork`, which is only called by the delegate in the `Page_Load` event. Anonymous methods now enable you to avoid creating a separate method and allow you to place the method directly in the delegate declaration instead. An example of the use of anonymous methods is shown in Listing 15-11.

Listing 15-11: Using delegates with an anonymous method

```
public partial class Default_aspx
{
    void Page_Load(object sender, EventArgs e)
    {
        this.Button1.Click += delegate(object myDelSender, EventArgs myDelEventArgs)
        {
            Label1.Text = "Welcome to the camp, I guess you all know why you're here.";
        };
    }
}
```

Using anonymous methods, you don't create a separate method. Instead you place necessary code directly after the delegate declaration. The statements and steps to be executed by the delegate are placed between curly braces and closed with a semicolon.

Operator Overloading

Operator overloading enables you to define the `+`, `-`, `*`, `/` and other operators in your classes just as the system classes can. This is a feature that has always been present in C#, but is now available in Visual Basic 8.0 as well. It gives you the capability to provide the objects in your classes with a similar feel when used with operators as if they were simply of type `String` or `Integer`.

Giving your classes this extended capability is a matter of simply creating a new method using the `Operator` keyword followed by the operator that you want to overload. An example of the `Operator` functions is illustrated in Listing 15-12.

Listing 15-12: Example operator overloading functions

```
Public Shared Operator +(ByVal Left As Point, ByVal Right As Size) As Point
    Return New Point(Left.X + Right.Width, Left.Y + Right.Height)
End Operator

Public Shared Operator -(ByVal Left As Point, ByVal Right As Size) As Point
    Return New Point(Left.X - Right.Width, Left.Y - Right.Height)
End Operator
```

Two different types of operators can be overloaded from Visual Basic — unary and binary operators:

- ❑ **Overloadable unary operators include:** `+` `-` `Not` `IsTrue` `IsFalse` `Widening` `Narrowing`
- ❑ **Overloadable binary operators include:** `+` `-` `*` `/` `\` `&` `Like` `Mod` `And` `Or` `Xor` `^` `<<` `>>` `=` `<>` `>` `<` `>=` `<=`

Partial Classes

Partial classes are a new feature included with the .NET Framework 2.0 and available to both C# and Visual Basic 8.0. These classes allow you to divide up a single class into multiple class files, which are later combined into a single class when compiled.

Partial classes are the secret of how ASP.NET keeps the new code-behind model simple. In ASP.NET 1.0/1.1, the code-behind model included quite a bit of code labeled as machine-generated code (code generated by the designer) and hidden within `#REGION` tags. Now, however, the code-behind file for ASP.NET 2.0 looks rather simple. A sample of the new code-behind model that uses partial classes is shown in Listing 15-13.

Listing 15-13: The new code-behind model using partial classes

```
VB
Imports Microsoft.VisualBasic

Namespace ASP

    Partial Class Default_aspx
```

```
Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    Label1.Text = "Hello " & Textbox1.Text
End Sub
End Class

End Namespace
```

C#

```
using System;

namespace ASP {

    public partial class Default_aspx
    {
        void Button1_Click (object sender, System.EventArgs e)
        {
            Label1.Text = "Hello " + Textbox1.Text;
        }
    }
}
```

This code-behind file contains a simple button-click event and nothing else. If you compare it to the designer code (as it was called) from the code-behind files found in ASP.NET 1.0/1.1, you notice a big difference between the two. What happened to all that code in the original code-behind file? It is still there, but now with the use of partial classes, all that necessary (but untouchable) code is kept in a separate class file. Upon compilation, the class file shown in Listing 15-14 is merged with the other class file. The result shows you that the code-behind files in ASP.NET 2.0 can consist simply of objects that you actually work with.

Partial classes are created with the use of the `Partial` keyword in Visual Basic and with the `partial` keyword in C# for any classes that are to be joined with a different class. The `Partial` keyword precedes the `Class` keyword for the classes to be combined with the original class. Besides using partial classes with every code-behind page that you work with in ASP.NET 2.0, you can also employ the same techniques with your own class files. You can associate two or more classes as part of the same class by using the procedure shown in Listings 15-14 and 15-15.

Listing 15-14: The first class

VB

```
Public Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer)
        Return (a + b)
    End Function
End Class
```

(continued)

Listing 15-14: *(continued)***C#**

```
public class Calculator
{
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

Listing 15-15: The second class**VB**

```
Partial Class Calculator
    Public Function Subtract(ByVal a As Integer, ByVal b As Integer)
        Return (a - b)
    End Function
End Class
```

C#

```
public partial class Calculator
{
    public int Subtract(int a, int b)
    {
        return a - b;
    }
}
```

When the two separate files are compiled, the two class files appear as a single object. The first class shown in Listing 15-15 is constructed just as a normal class is, whereas any additional classes that are to be made a part of this original class use the new `Partial` keyword. A consumer using the compiled `Calculator` class will see no difference. After the consumer of the `Calculator` class creates an instance of this class, this single instance has both an `Add` and a `Subtract` method to it. This is illustrated in Figure 15-1.



Figure 15-1

Visual Basic XML Documentation

Like C#, Visual Basic 8.0 now includes the capability to create XML documentation from comments that are left in your VB files. Visual Basic denotes XML documentation remarks in code with the use of three successive single quotation marks (''''). This is similar to how C# does it. C# uses three forward slashes (///). Comments left in VB code can then be converted to documentation. Listing 15-16 shows the use of XML documentation in code.

Listing 15-16: Visual Basic code with comments for XML documentation

```
Imports Microsoft.VisualBasic

''' <summary>My Calculator Class</summary>
Public Class Class1

    ''' <summary>This Add method returns the value of two numbers
    '''   added together</summary>
    ''' <param name="a">First number of the collection of numbers to
    '''   be added</param>
    ''' <param name="b">Second number of the collection of numbers to
    '''   be added</param>
    Public Function Add(ByVal a As Integer, ByVal b As Integer)
        Return (a + b)
    End Function

End Class
```

The Visual Basic 8.0 compiler now includes a new `/doc` command that is similar to the way C# works with XML documentation. Compiling your VB code using the `/doc` command causes the compiler to produce the XML documentation with the compilation.

New Visual Basic Keywords

Visual Basic 8.0 introduces a couple of new keywords that can be utilized in your ASP.NET 2.0 applications. The keywords were brought to the language to make it easier to perform some common tasks, such as working in loops or destroying resources as early as possible. Look at a couple of the new additions to the Visual Basic language.

Continue

The `Continue` statement is an outstanding new addition to the Visual Basic language that was brought on board to enable you to work through loops more logically in some specific situations. When working in a loop, it is sometimes beneficial to stop the conditional flow and move onto the next item in the collection if the item being examined simply doesn't fit your criteria. This logic can now be implemented better because of the new `Continue` statement. Listing 15-17 shows an example of the use of the `Continue` statement.

Listing 15-17: Using the Continue statement

```
Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim myString As String
    Dim count As Integer = 0
    myString = "The St. Louis Rams will go to the Superbowl this year."

    For i As Integer = 0 To (myString.Length() - 1)
        If (myString(i).Equals(" c")) Then Continue For
        count += 1
    End For
End Sub
```

```
Next

    Label1.Text = "There are " & count.ToString() & _
        " characters used (minus spaces)."
```

```
End Sub
```

This little example takes a `String` and counts each character in the complete string that is not a space. If a space is encountered, the `Continue` statement finds this in the check and immediately stops execution of the loop for that particular item in the collection. It then hands over the execution of the loop to the next item in the collection. In this example, you could easily check for the characters with a nested `If` statement, but using multiple nested `If` statements can get confusing sometimes. The use of the `Continue` statement makes the logic contained within the `For` loop very evident and clean.

The `Continue` statement is not only meant to be used within a `For` loop, but you can also use this new keyword with other language features that loop through a collection of items — such as the `Do` and `While` statements. The following example shows how to use the `Continue` statement with the four available options:

```
For [statement]
    ...

    If [statement] Then Continue For

    ...
Next

For Each [statement]
    ...

    If [statement] Then Continue For

    ...
Next

Do While [statement]
    ...

    If [statement] Then Continue Do

    ...
Loop

While [statement]
    ...

    If [statement] Then Continue While

    ...
End While
```

As you can see, you have many ways to use this new keyword to make your code easier to read and manage.

Using

Although the `using` keyword in C# is quite prevalent (one use being to import namespaces into a class), the new Visual Basic `Using` keyword should not be confused with this C# version. It is similar to the C# `using` statement that defines the scope of an object.

You use the `Using` keyword in Visual Basic to ensure that expensive resources get destroyed as soon as possible instead of allowing them to sit in memory until the method is executed. You can now destroy expensive resources, such as connection objects and COM wrappers, immediately when you have finished using them instead of waiting for the garbage collector to come by and make its rounds. An example of working with the `Using` keyword is illustrated in Listing 15-18.

Listing 15-18: Working with the `Using` keyword

```
Using myConn As New SqlConnection
    ' Work with the SqlConnection object
End Using
```

In Listing 15-18, you can see that instead of using the `Dim` keyword to create a new instance of the `SqlConnection` object, the `Using` keyword is used in its place. If you utilize the `Using` keyword, you must close the `Using` statement with an `End Using` statement. The `End Using` statement is located at the point where the `SqlConnection` object is destroyed from memory.

My

My, oh my, what a new keyword! The `My` keyword is a novel concept to quickly give you access to your application, the computer, or the network in which the application resides. The `My` keyword has been referred to as a way of speed-dialing common but complicated resources that you need access to. Using the `My` keyword, you can quickly get access to a wide variety of items such as user details or specific settings of the requestor's browser.

If you type the `My` keyword into your application, you will notice that IntelliSense provides you with three items to work with: `Application`, `Computer`, and `User`. Although this new keyword works best in the Windows Forms environment, there are still things you can use in the Web Forms world.

If you want to get at the identity of a user, for example, you can use the following construct:

```
Label1.Text = My.User.Identity.Name.ToString()
```

Another example is checking whether the browser making the request to the application is a mobile device. For this, you would use the following construction:

```
Label1.Text = My.Application.Request.Browser.IsMobileDevice.ToString()
```

You can get to the information stored in files, system settings, and more in a number of quick ways. The best way to explore the `My` namespace is to look through IntelliSense and see what is available.

Global

The `Global` keyword was added as a top-root namespace to avoid any namespace conflicts that might arise from similarly named namespaces. Here is an example of how you can use the `Global` keyword:

```
Global.System.String
```

Summary

This short chapter looked at *some* of the changes to the C# and Visual Basic languages in their latest releases. When a new version of the .NET Framework comes along, everything associated with it is refreshed at the same time. With .NET Framework 2.0, not only do you get a new version of ASP.NET, but new versions of all the Microsoft .NET-compliant languages, Windows Forms, and more.

The new features of C# and VB illustrated in this chapter can be directly used in the ASP.NET 2.0 applications you are building — giving you applications that contain cleaner and better-performing code.

Index

> (greater than), PathSeparator property, 132
| (pipe character), SiteMapPath control, 132

A

accelerator keys, Label control and, 369
Access, personalization provider, 281-282
AccessDataSource control, 96-109
Add Web Reference dialog, 37
administration overview, 6-7
AdRotator server control, 376-380
Advanced tab, MMC ASP.NET tab, 397-399
advertisements, AdRotator control, 377
AllowReturn attribute, Wizard server control, 357
anonymous identification events, 278-279
anonymous methods
 C#, 421
 Visual Basic, 421
anonymous personalization, 275-278
 anonymous identification events, 278-279
 cookies, 276-277
 identifier storage, 277
 options, 279
anonymous users, migrating, 279-280
Application settings option, Authentication tab (MMC), 394

Application tab

MMC ASP.NET tab, 394-395
 WAT, 404

applications

configuration, cache invalidation, SQL Server, 331-332
 folders
 Code folder, 61-65
 Resources folder, 66-69
 Themes folder, 66
 location
 FTP, 44
 IIS, 43-44
 Web server, 41-43
 registration, users, 235-236
 roles
 adding users, 256
 adding/retrieving, 252-255
 all user's roles, displaying, 256-259
 checking users in roles, 259-260
 deleting, 255
 removing users, 259
 themes, 205-208
<asp> element
 <CheckBoxField>, 93
 <HyperLinkField>, 94
 <TemplateField>, 95

ASP.NET history, 2

ASP.NET tab, MMC, 387

- Advanced tab, 397-399
- Application tab, 394-395
- Authentication tab, 393-394
- Authorization tab, 391-392
- Custom Errors, 390-391
- General tab, 389-390
- State Management tab, 395-396

ASP.NET Whidbey, 2

AspNetAccessProvider user registration information, 226

<asp.Wizard> element, 355

attributes

- page directives, 51
- siteMapNode element, XML, 129
- Wizard server control
 - AllowReturn, 357
 - StepType, 357-358

authentication

- definition, 226
- forms authentication, 225
- users
 - adding, 229-235
 - LoginName server control and, 241
 - LoginStatus server control and, 240-241
 - number online, displaying, 242-243
- Web site membership, 236
 - <authorization> element, 236
 - login programmatically, 239-240
 - Login server control, 237-239
 - setup, 226-229
- web.config file, 227
 - <forms> element, 228

<authentication> element, web.config file, 227

Authentication tab, MMC ASP.NET tab, 393-394

authorization

- definition, 226
- LoginView control and, 248-249

<authorization> element, 236

Authorization tab, MMC ASP.NET tab, 391-392

B

banner ads, AdRotator control, 377

beta releases of software, 1

boxing, 416

breadcrumb navigation, 130

built-in styles, TreeView control, 138-139

BulletedList server control, 341-346

Button control, 371-372

C

C#

- changes overview, 413-414
- classes, partial classes, 422-424
- generics, 414-418
- methods, anonymous, 421
- operator overloading, 422

Cache object

- data caching and, 324
- dependencies, 324
- SQL Server cache dependencies, 335-338

CacheDependency class, 325

caching

- data caching, 324
- dependencies, 324
 - Cache object, 335-338
 - Request object, 334-335
 - SQL Server, 325-330
- invalidation
 - configuring applications, 331-332
 - SQL Server databases, 326-330
 - SQL Server tables, 327-329
 - tables, adding, 334
 - testing, 332-338
- master pages and, 201
- output caching, 323
- partial page caching, 324
- SQL cache invalidation, 7
- SQL Server Enterprise Manager, 327-328

casting, boxing, 416

ChangePassword server control, 244-245

CheckBoxList server control, 372-373

child nodes, 139

Choose Location dialog, 42

classes

C#, partial classes, 422-424

CacheDependency, 325

generics, 417

Portal Framework and, 317-321

WebPart, 320-321

WebPartManager, 317-318

WebPartZone, 319-320

SiteMap, 128, 168-170

Visual Basic, partial classes, 422-424

code

compiling, 15

content pages, 180-193

generating device-specific, 8-9

inline coding, 47-48

master pages, 177-180

code-behind model, 49-51, 422

code-behind pages, 47

**code-change notification, Document Window
(Visual Studio), 26**

\Code folder, 61-65

collections, generics and, 415

columns, GridView control, 93-96

<asp:CheckBoxField>, 93

<asp:HyperLinkField>, 94

<asp:TemplateField>, 95

bound, 83-87

compilation, 69-73

in-place, 70

precompilation for deployment, 71

compiling code, 15

connection strings, 123-124

container-specific master pages, 199-200

content pages

code, 180-193

controls, 187-193

default content, specifying, 194-195

event order, 200-201

languages, 184-185

master pages and, 176, 186

page types, mixing, 184-185

properties, 187-193

title, 186

Continue keyword, Visual Basic, 426-427

control sections, Toolbox (Visual Studio), 29-30

controls

AccessDataSource, 96-109

AdRotator, 376-380

BulletedList, 341-346

Button, 371-372

ChangePassword, 244-245

CheckBoxList, 372-373

CreateUserWizard, 229-233

custom, themes and, 221-222

data source controls, 75-76

data-bound server controls, 76-77

DataGrid, 18

DataSetDataSource, 117

DetailsView, 96-109

DropDownList, 372-373

DynamicImage, 14, 361-365

FileUpload, 348-351

GridView, 77

bound columns and, 83-87

columns and, 93-96

data deletion, 92-93

paging, 79-82

reading data, 77-78

row editing, 87-92

row sorting, 82

HiddenField, 346-348

Image, 374

ImageButton, 371-372

ImageMap, 366-368

Label, changes in, 369-371

LinkButton, 371-372

ListBox, 372-373

Literal, 376

controls (continued)

- Login, 237-239
- LoginName, 241
- LoginStatus, 240-241
- LoginView, 248-249
- master pages, 187-193
- Menu, 156-157
 - binding to XML file, 163-165
 - events, 163
 - images as dividers in menus, 162
 - item layout, 160-161
 - pop-out symbol, 161
 - pre-defined styles, 157-158
 - styles, dynamic items, 159
 - styles, static items, 158-159
- MultiView, 351-353
- new, 18
- ObjectDataSource, 114-116
- Panel, 380-382
- PasswordRecovery, 245-247
- Portal Framework, adding to, 312
- RadioButtonList, 372-373
- SiteMapDataSource, 116, 165
 - SiteMapViewType property, 165
 - StartingNodeType property, 166-167
- SiteMapPath, 13, 130-132
 - child elements, 135
 - ParentLevelsDisplayed property, 134-135
 - PathDirection property, 134
 - PathSeparator property, 132-134
 - ShowToolTips property, 135
- skins, assigning SkinID programmatically, 220
- SqlDataSource, 18, 77
- Table, 374-376
- TreeView, 13, 136-138
 - binding to XML file, 140-142
 - built-in styles, 138-139
 - icons, custom, 145-147
 - node connection, 147-149
 - nodes, 139
 - options, multiple, 142-145

- programmatical manipulation, 150-156
- ShowCheckBoxes property, 142
- validation controls, 382-385
- View, 351-353
- WebPartManager, 293-294
- WebPartPageMenu, 301-309
- WebPartZone, 298
 - default elements, 299-300
 - LayoutOrientation attribute, 299
 - <ZoneTemplate> element, 299
- Wizard, 355-361
- XmlDataSource, 109-114
- cookies, anonymous identification and, 276-277**
- counter system, WAT, 407-410**
- CreateUserWizard server control, 232-233**
- cross-page posting, 54-60**
- CSS (Cascading Style Sheets)**
 - files, themes and, 211-214
 - themes and, 203
- Custom Errors tab, MMC ASP.NET tab, 390-391**

D

- data caching, 324**
- data source controls, 75-76**
 - Visual Studio 2005 and, 118-122
- data-bound server controls, 76-77**
- databases, SQL Server**
 - disabling for cache invalidation, 330
 - enabling for cache invalidation, 326-327
- DataGrid control, 18**
- DataSetDataSource control, 117**
- delegates**
 - anonymous methods, 421
 - generics, 417
- deleting data, GridView control, 92-93**
- dependencies**
 - caching, 324
 - Cache object, 335-338
 - Request object, 334-335
 - SQL Server, 325-330

Design tab, Document Window (Visual Studio), 24

DetailsView control, 96-109

- field display, 101-103
- GridView control and, 103-105
- row manipulation, 105-109

developers

- new infrastructures, 9-14
- productivity and, 3-6

device-specific code generation, 8-9

directives, 51-52

Document Window, Visual Studio, 23

- code-change notification system, 26
- Design tab, 24
- error notifications, 27-28
- page tabs, 25-26
- Source tab, 24
- tag navigator, 25

documentation, Visual Basic, 425-426

DropDownList server control, 372-373

Duration attribute, output caching and, 324

dynamic items, styles, 159

DynamicImage control, 14

DynamicImage server control, 361-365

E

Edit/Add Connection String dialog, ASP.NET tab, MMC, 390

elements

- child elements, SiteMapPath control, 135
- XML
 - siteMap, 129
 - siteMapNode, 129

error notification, Document Window (Visual Studio), 27-28

events

- anonymous identification events, 278-279
- content page order, 200-201
- master page order, 200-201
- Menu control, 163

pages, 53-54

Wizard server control, 360-361

Extensions, Front Page, 45

F

fields

- displaying, DetailsView control and, 101-103
- hidden, HiddenField server control, 346-348

files, uploading with FileUpload server control, 348-351

FileUpload server control, 348-351

folders

- applications
 - Code folder, 61-65
 - Resources folder, 66-69
 - Themes folder, 66
- themes
 - file structure, 208-209
 - skins, 209-211

foreach loop, iterations, 419

Forms Authentication, 225

<forms> element, web.config file, 228

FrontPage, Extensions, 45

FTP, application location, 44

G

General tab, MMC ASP.NET tab, 389-390

generating code, device-specific, 8-9

GenericReturn method, 418

generics

- C#, 414-418
- classes, 417
- collections and, 415
- delegates, 417
- methods, 417
- Stack class, 415
- Visual Basic, 414-418

GetEnumerator function, iterations and, 419

Global keyword, Visual Basic, 429

greater than sign (>), PathSeparator property, 132

GridView control, 77

bound columns and, 83-87

columns and, 93-96

<asp:CheckBoxField>, 93

<asp:HyperLinkField>, 94

<asp:TemplateField>, 95

data deletion, 92-93

DetailsView control and, 103-105

paging, 79-82

reading data, 77-78

row editing, 87-92

row sorting, 82

grouped personalization properties, 271

H

headers, Wizard server control, 358

HiddenField server control, 346-348

Home tab, WAT, 401

hotkeys, Label control and, 369

I

IBuySpy, 10

icons, TreeView control, 145-147

identifiers, storage, 277

IIS, application location, 43-44

image maps, ImageMap server control, 366-368

Image server control, 374

ImageButton control, 371-372

ImageMap server control, 366-368

images

disk, 361

DynamicImage server control, 361-365

generating, 14

menu item separation, Menu control, 162

resizing, 363

streaming, 364-365

themes, 214-217

importing Visual Studio settings, 38-40

in-place precompilation, 70

inline coding, 47-48

integers, Stack class, 416

iterators

foreach loop, 419

GetEnumerator function, 419

K

keywords, Visual Basic

Continue, 426-427

Global, 429

My, 428

Partial, 423

Using, 428

L

Label control

accelerator keys, 369

AccessKey attribute, 370

changes, 369-371

hotkeys, 369

languages, content pages, 184-185

LayoutOrientation attribute, WebPartZone

control, 299

leaf nodes, 139

LinkButton control, 371-372

ListBox server control, 372-373

**lists, bulleted lists (BulletedList server control),
341-346**

Literal server control, 376

logging in programmatically, 239-240

Login server control, 237-239

LoginName server control, 241

LoginStatus server control, 240-241

LoginView server control, 248-249

loops, foreach, 419

lost windows in Visual Studio, 34

M

machine.config file

- editing contents, 6
- <roleManager> section, 250

management overview, 6-7

master pages, 16-17

- caching and, 201
- coding, 177-180
- container-specific, 199-200
- content pages, 176
 - code, 180-193
 - default, specifying, 194-195
 - specifying, 186
- controls, 187-193
- event order, 200-201
- need for, 173-175
- nesting, 196-198
- properties, 187-193
- sample code, 178
- specifying, 186
- subpages, 176

Membership and Role Management Service, 225

Membership option, Authentication tab (MMC), 394

membership to Web site, 236

- <authorization> element, 236
- Login server control, 237-239
- log in users programmatically, 239-240
- setup, 226-229

Menu control, 156-157

- events, 163
- images as dividers in menus, 162
- item layout, 160-161
- pop-out symbol, 161
- styles
 - dynamic items, 159
 - pre-defined, 157-158
 - static items, 158-159
- XML files, binding to, 163-165

methods

- anonymous
 - C#, 421
 - Visual Basic, 421
- GenericReturn, 418
- generics, 417

migration, anonymous users, 279-280

MMC (Microsoft Management Console), 6, 387

- ASP.NET tab, 389
- Advanced tab, 397-399
- Application tab, 394-395
- Authentication tab, 393-394
- Authorization tab, 391-392
- Custom Errors tab, 390-391
- General tab, 389-390
- Site Management tab, 395-396

MultiView server control, 351-353

My keyword, Visual Basic, 428

N

navigation

- breadcrumb navigation, 130
- SiteMapPath control and, 130
- tag navigator, Document Window (Visual Studio), 25
- Wizard server control, 357-360

nesting master pages, 196-198

new features, infrastructures, 9-14

nodes, TreeView control, 139

- adding, 153-156
- connection, 147-149
- expanding/collapsing programmatically, 150-153

O

ObjectDataSource control, 114-116

object references, 35-37

operators, overloading

- C#, 422
- Visual Basic, 422

options, TreeView control, 142-145

ordering events

content pages, 200-201

master pages, 200-201

output caching, 323

OutputCache page directive, 324

overloading operators

C#, 422

Visual Basic, 422

P

page directives, OutputCache, 324

page tabs, Document Window (Visual Studio), 25-26

page title, 186

pages

code-behind, 47

cross-page posting, 54-60

directives, 51-52

events, 53-54

master pages, 16-17

structure, 45-51

themes, 17, 220

Web Parts, adding, 302-304

paging, GridView control, 79-82

Panel server control, 380-382

parent node, 139

ParentLevelsDisplayed property, SiteMapPath control, 134-135

partial classes

code-behind model and, 422

Visual Basic/C#, 422-424

Partial keyword, 423

partial page caching, 324

PasswordRecovery server control, 245-247

passwords

ChangePassword server control, 244-245

PasswordRecovery server control, 245-247

PathDirection property, SiteMapPath control, 134

PathSeparator property, SiteMapPath control, 132-134

performance overview, 7-8

personalization

providers

Access, 281-282

multiple, 289-290

SQL, 282-288

SQL scripts, 286-288

SQL Server Setup Wizard, 282-286

users and, 10

Personalization model, 263-264

personalization properties

anonymous personalization, 275-278

anonymous identification events, 278-279

cookies and, 276-277

identifier storage, 277

options, 279

creating, 265-266

default values, 275

grouped, uses, 271

groups, 270-271

types, 271-272

custom, 272-274

uses, 266-269

pipe character (|), SiteMapPath control, 132

pop-out symbol, Menu control and, 161

Portal Framework, 10

classes, 317-321

WebPart class, 320-321

WebPartManager class, 317-318

WebPartZone class, 319-320

controls, adding, 312

modes, 292

Web Parts, 292

adding to pages, 302-304

connecting, 309

moving, 305-306

settings, editing, 306-309

verbs, 314-317

WebPartManager control, 293-294
 WebPartPageMenu control, 301-309
 customizing, 309
 WebPartZone control, 298
 default elements, 299-300
 LayoutOrientation attribute, 299
 <ZoneTemplate> element, 299
 zones, modification capability, 310-311
posting, cross-page posting, 54-60
pre-defined styles, Menu control, 157-158
precompilation for deployment, 71
productivity, developers and, 3-6
Profile API, 264
Profile tab, WAT, 403-404
projects, Visual Studio, 35
properties
 master page, 187-193
 ParentLevelsDisplayed (SiteMapPath control),
 134-135
 PathDirection (SiteMapPath control), 134
 PathSeparator (SiteMapPath control), 132-134
 personalization
 anonymous, 275-279
 creating, 265-266
 custom types, 272-274
 default values, 275
 grouped, 271
 groups, 270-271
 types, 271-272
 uses, 266-269
 ShowCheckBoxes (TreeView control), 142
 ShowToolTips (SiteMapPath control), 135
 SiteMapViewType, SiteMapDataSource
 control, 165
 StartingNodeType, SiteMapDataSource
 control, 166-167
 ValidationGroup, 383
Properties Window, Visual Studio, 33
Property Pages dialog, 35-36
Provider tab, WAT, 405-406

R

RadioButtonList server control, 372-373
references to objects, 35-37
registering users, 235-236
**Request object, SQL Server cache dependencies,
 334-335**
resizing images, DynamicImage control, 363
\Resources folder, 66-69
role management
 adding users, 256
 adding/retrieving roles, 252-255
 all user's roles, displaying, 256-259
 checking users in roles, 259-260
 deleting roles, 255
 removing users, 259
 Web site setup, 249
 adding users to roles, 256
 adding/retrieving application roles, 252-255
 all user's roles, displaying, 256-259
 checking users in roles, 259-260
 deleting roles, 255
 removing users, 259
 <roleManager> section of config file, 250
 web.config file edits, 252
<roleManager> section, configuration file, 250
Roles option, Authentication tab (MMC), 394
root node, 139
rows
 DetailsView control, 105-109
 GridView control,
 editing, 87-92
 sorting, 82

S

scalability overview, 7-8
scrolling, Panel control and, 380
Security tab, WAT, 402-403
server controls. See also controls
 AdRotator, 376-380
 BulletedList, 341-346

server controls (continued)

- Button, 371-372
- ChangePassword, 244-245
- CheckBoxList, 372-373
- counter system, 409
- CreateUserWizard, 229-233
- data-bound server controls, 76-77
- DropDownList, 372-373
- DynamicImage, 361-365
- FileUpload, 348-351
- HiddenField, 346-348
- Image, 374
- ImageButton, 371-372
- ImageMap, 366-368
- Label, changes in, 369-371
- LinkButton, 371-372
- ListBox, 372-373
- Literal, 376
- Login, 237-239
- LoginView, 248-249
- MultiView, 351-353
- new, 18
- Panel, 380-382
- PasswordRecovery, 245-247
- RadioButtonList, 372-373
- Table, 374-376
- themes, removing from, 206-207
- validation controls, 382-385
- View, 351-353
- Wizard, 355-361

Server Explorer, Visual Studio, 33

ShowCheckBoxes property, TreeView control, 142

ShowToolTips property, SiteMapPath control, 135

site maps, 128

SiteMap class, 128, 168-170

siteMap element, XML, 129

SiteMapDataSource control, 116, 165

- SiteMapViewType property, 165
- StartingNodeType property, 166-167

siteMapNode element, XML, 129

SiteMapPath control, 13, 130-132

- | (pipe character), 132
- child elements, 135
- navigation, breadcrumb navigation, 130
- ParentLevelsDisplayed property, 134-135
- PathDirection property, 134
- PathSeparator property, 132-134
- ShowToolTips property, 135

SiteMapViewType property, SiteMapDataSource control, 165

skins, 209-211

- multiple, 218-220
- SkinID, assigning programmatically, 220

smart tags, Visual Studio, 37

<Smtip> element, PasswordRecovery control and, 246

Solution Explorer, Visual Studio

- introduction, 31-33
- toolbar, 31

source controls, data source controls, 75-76

Source tab, Document Window (Visual Studio), 24

SQL (Structured Query Language)

- personalization provider, 282-288
- scripts, personalization and, 286-288

SQL cache invalidation, 7

SQL Server

- cache dependency, 325-330
 - databases, disabling for cache invalidation, 330
 - databases, enabling for cache invalidation, 326-327
- tables, disabling for cache invalidation, 329-330
- tables, enabling for cache invalidation, 327-329
- cache invalidation
 - application configuration, 331-332
 - tables, adding, 334
 - testing, 332-338
- caching
 - Cache object, 335-338
 - Enterprise Manager and, 327-328
 - Request object, 334-335

SQL Server Setup Wizard, 282-286

SqlDataSource control, 18, 77

Stack class

- generic version, 415
- integers and, 416

StartingNodeType property, SiteMapDataSource control, 166-167

State Management tab, MMC ASP.NET tab, 395-396

static items, styles (Menu control), 158-159

StepType attribute, Wizard server control, 357-358

storage, users, 231

streaming images, DynamicImage server control, 364-365

strings, connection strings, 123-124

styles

- built-in, TreeView control, 138-139
- Menu control
 - dynamic items, 159
 - pre-defined, 157-158
 - static items, 158-159

subpages, master pages and, 176

T

Table server control, 374-376

tables, SQL Server

- cache invalidation, 334
- disabling for cache invalidation, 329-330
- enabling for cache invalidation, 327-329

tag navigator, Document Window (Visual Studio), 25

tags, smart tags (Visual Studio), 37

themes, 17, 203

- applications, removing from, 208
- applying
 - to application, 205
 - to applications on server, 206
 - to page, 204-205
- assigning to pages programmatically, 220
- controls and, custom, 221-222

creating, folder structure, 208-209

CSS files, including, 211-214

images in, 214-217

removing from Web pages, 207

server controls, removing from, 206-207

skins, 209-211

\Themes folder, 66

title, page title, 186

tool tips, 135

Toolbox, Visual Studio, 29-30

TreeView control, 13, 136-138

- icons, custom, 145-147
- images, themes and, 215
- nodes, 139
 - adding, 153-156
 - connection, 147-149
 - expanding/collapsing programmatically, 150-153
- options, multiple, 142-145
- programmatical manipulation, 150-156
- ShowCheckBoxes property, 142
- styles, built-in, 138-139
- XML files, binding to, 140-142

U

uploading files, FileUpload server control and, 348-351

users

- adding, 229
 - CreateUserWizard server control, 229-231
 - programmatically, 233-235
 - storage, 231
- anonymous, migrating, 279-280
- authenticated
 - LoginName server control and, 241
 - LoginStatus server control and, 240-241
 - number online, displaying, 242-243
- log in, programmatical, 239-240
- membership, 9
- number online, displaying, 242-243

users (continued)

- personalization and, 10
- registration, 235-236
- roles, managing, 9

Using keyword, Visual Basic, 428

V

validation, SQL cache invalidation, 7

validation server controls, 382-385

ValidationGroup property, 383

verbs, Web Parts, 314-317

View server control, 351-353

Visual Basic

- changes overview, 413-414
- classes, partial classes, 422-424
- generics, 414-418
- keywords
 - Continue, 426-427
 - Global, 429
 - My, 428
 - Using, 428
- methods, anonymous, 421
- operator overloading, 422
- XML, documentation, 425-426

Visual Studio

- Document Window, 23-28
- importing settings, 38-40
- lost windows, 34
- objects, references, 35-37
- projects, creating, 35
- Properties Window, 33
- saving settings, 38-40
- Server Explorer, 33
- smart tags, 37
- Solution Explorer, 31-33
- Toolbox, 29-30

Visual Studio 2005, 19-20

- data source controls and, 118-122

W

WAT (Web Site Administration Tool), 399

- accessing, 400
- Application tab, 404
- counter system, 407-410
- Home tab, 401
- Profile tab, 403-404
- Provider tab, 405-406
- Security tab, 402-403

Web Parts

- adding to pages, 302-304
- connecting, 309
- moving, 305-306
- overview, 291-292
- settings, editing, 306-309
- verbs, 314-317

Web server, application location, 41-43

Web site, role management setup, 249

- adding users to roles, 256
- adding/retrieving application roles, 252-255
- all user's roles, displaying, 256-259
- checking users in roles, 259-260
- deleting roles, 255
- removing users, 259
- <roleManager> section of config file, 250
- web.config file edits, 252

Web Site Administration Tool, 261

web.config file

- <authentication> element, 227
- counter system, 408
- editing, 252
- editing contents, 6
- <forms> element, 228
- personalization properties, 265
 - groups, 270-271
- <roleManager> section, 250
- SQL Server cache and, 331

web.sitemap file example, 128

WebPart class, 320-321

WebPartManager class, 317-318

WebPartManager control, 293-294

zone layouts, 294-298

WebPartPageMenu control, 301-309

WebPartZone class, 319-320

WebPartZone control, 298

elements, default, 299-300

LayoutOrientation attribute, 299

<ZoneTemplate> element, 299

Whidbey, 2

windows

Design Window

page tabs, 25-26

tag navigator, 25

Document Window

code-change notification system, 26

error notification, 27-28

Document Window, Visual Studio, 23-24

lost windows in Visual Studio, 34

Properties Window (Visual Studio), 33

Wizard server control, 355-361

<asp.Wizard> element, 355

attributes

AllowReturn, 357

StepType, 357-358

events, 360-361

headers, 358

navigation and, 359-360

side navigation, 357

X-Y-Z

XML (eXtensible Markup Language)

documentation, Visual Basic, 425-426

elements

siteMap, 129

siteMapNode, 129

files

binding Menu control to, 163-165

binding TreeView control to, 140-142

XML advertisement files, AdRotator server control, 377

XmlDataSource control, 109-114

zones

Portal Framework, modification capability, 310-311

WebPartManager control, layout, 294-298

<ZoneTemplate> element, WebPartZone control, 299