

Michael Amundsen
Paul Litwin

ASP.NET for Developers

SAMS

ASP.NET for Developers

Copyright © 2002 by Sams Publishing

All rights reserved. No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from the publisher. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions. Nor is any liability assumed for damages resulting from the use of the information contained herein.

International Standard Book Number: 0-672-32038-x

Library of Congress Catalog Card Number: 00-105163

Printed in the United States of America

First Printing: December 2001

04 03 02 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Active Server Pages is a trademark of Microsoft Corporation.

ASP.NET is a trademark of Microsoft Corporation.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book or from the use of programs accompanying it.

ASSOCIATE PUBLISHER

Linda Engelman

ACQUISITIONS EDITOR

Neil Rowe

DEVELOPMENT EDITORS

Laurie McGuire

Kevin Howard

MANAGING EDITOR

Charlotte Clapp

PROJECT EDITOR

Elizabeth Finney

COPY EDITOR

Ned Snell

INDEXER

Eric Schroeder

PROOFREADER

Kelly Ramsey

TECHNICAL EDITOR

Ralph Arvesen

TEAM COORDINATOR

Lynne Williams

MEDIA DEVELOPER

Dan Scherf

INTERIOR DESIGNER

Karen Ruggles

COVER DESIGNER

Aren Howell

PAGE LAYOUT

Tricia Bronkella

Rebecca Harmon

Brad Lenser

Cheryl Lynch

Michelle Mitchell

Overview

Introduction 1

PART I UNDERSTANDING THE .NET ENVIRONMENT 7

- 1 Introducing .NET 9
- 2 Understanding the .NET Framework 17
- 3 What's New with ASP.NET? 25

PART II VISUAL BASIC .NET QUICKSTART 37

- 4 Understanding Visual Basic .NET Syntax and Structure 39
- 5 Working with Numbers, Strings, Dates, and Arrays in Visual Basic .NET 67
- 6 Using Namespaces and Assemblies with Visual Basic .NET 97

PART III BUILDING WEB PAGES WITH ASP.NET 107

- 7 Understanding ASP.NET Web Forms 109
- 8 Creating Simple Web Pages with the HTML Server Controls 127
- 9 Creating Interactive Forms with Web Form Server Controls 149
- 10 Designing Advanced Interfaces with Web Form List Controls 181
- 11 Improving Your User Interfaces with Validation Controls 215
- 12 Adding User Controls to Your Web Forms 231

PART IV HANDLING DATA ACCESS WITH ADO.NET 243

- 13 Introduction to ADO.NET and Data Binding 245
- 14 Accessing Data with .NET Data Providers 259
- 15 Working with ADO.NET DataSets 283

PART V CREATING AND USING WEB SERVICES 327

- 16 Understanding the Web Service Model 329
- 17 Publishing Web Services with ASP.NET 341
- 18 Consuming Web Services with ASP.NET 357

PART VI CONFIGURING AND DEPLOYING ASP.NET SOLUTIONS 371

- 19 Tracing, Debugging, and Optimizing Your ASP.NET Applications 373
- 20 Implementing ASP.NET Security 393
- 21 Configuring and Deploying Your ASP.NET Solutions 409

APPENDIX

- A Moving to VB .NET from VB6 or VBScript 425

INDEX 433

Contents

INTRODUCTION	1
PART I UNDERSTANDING THE .NET ENVIRONMENT 7	
1 INTRODUCING .NET 9	
The New Face of Microsoft Web Development	10
The Move from Workstation to Distributed Computing	10
Internet As a Key Mode of Delivery	10
Operating System and Development Tools Lead the Way	11
The Power of the .NET Platform	12
An OS-Neutral Environment	12
Device Independent	13
Wide Language Support	13
Internet-Based Component Services	14
Summary	15
2 UNDERSTANDING THE .NET FRAMEWORK 17	
The Common Language Runtime (CLR)	18
Code Management and Execution	18
Security Support	19
The Common Language Runtime (CLR)	19
Error Handling and Garbage Collection	20
The .NET Framework Class Libraries	21
System Classes	21
Data and XML Classes	22
Windows Forms and Drawing Classes	23
Web Classes	23
Summary	24
3 WHAT'S NEW WITH ASP.NET? 25	
ASP.NET Web Forms and Web Services	25
Improved Web Page Authoring	25
ASP.NET Server Controls	28
ASP.NET Web Services	32
ASP.NET Infrastructure	33
Powerful Security Model	33
Improved Deployment and Update	33
Easier Configuration	34
Increased Scalability and Availability	35
Summary	36
PART II VISUAL BASIC .NET QUICKSTART 37	
4 UNDERSTANDING VISUAL BASIC .NET SYNTAX AND STRUCTURE 39	
The New Look of Visual Basic	40
Getting Started with Visual Basic .NET	41
Statements and Lines	41
Comments	41
Operators	41
Using Procedures	43
Subroutines	43
Functions	43

Using Variables and Parameters	44
Constants	46
Implicit and Explicit Variable Declarations	46
Option Explicit versus Option Strict	47
Arrays	47
Passing Parameters	49
Using Branching and Looping Structures	52
Branching in VB .NET	52
Looping in VB .NET	53
Creating Objects	57
OOP Primer	57
Creating a Class	58
Using Property Statements	59
Inheritance	62
Accessibility of Inherited Properties and Methods	63
Overriding Methods	64
Constructing an Object	65
Summary	66
5 WORKING WITH NUMBERS, STRINGS, DATES, AND ARRAYS IN VISUAL BASIC .NET	67
Using the System Classes	68
Instance and Static Class Members	68
Working with Numbers	69
E and PI	69
Trigonometry	69
Rounding and Truncation	69
Powers, Roots, and Logarithms	70
Miscellaneous Mathematical Functionality	72
Manipulating Strings	72
Determining a String's Length	73
Searching for and Extracting Substrings	73
Splitting and Joining Strings	76
Trimming and Padding	76
Comparing Strings	78
Changing the Case of a String	79
DateTime Arithmetic	79
Now and Today	79
Creating DateTime Values	80
Comparing Dates	80
Adding and Subtracting DateTime Values	80
Parsing DateTime Values	83
Converting Values	84
Formatting Values	86
Formatting Numbers	86
Formatting Dates	89
Managing Arrays	91
Determining the Boundaries of an Array	91
Sorting and Reversing the Elements of an Array	92
Searching an Array	94
Summary	95
6 USING NAMESPACES AND ASSEMBLIES WITH VISUAL BASIC .NET	97
Namespaces and Assemblies	97
Relating Namespaces and DLL Assemblies	98

Creating Assemblies	99
Defining a Namespace Area	99
Creating an Assembly Within a Namespace Definition	100
Creating an Assembly Without a Namespace Definition	102
Compiling an Assembly	102
Importing Assemblies	102
Using Imported Assemblies	103
Compiling with Imported Namespaces	105
Summary	105

PART III BUILDING WEB PAGES WITH ASP.NET 107

7 UNDERSTANDING ASP.NET WEB FORMS 109

Understanding the Web Forms Code Model	109
In-Page versus Code-Behind Format	110
The Web Form Object Life Cycle	114
Handling Client-Side Events on the Server	115
Web Form Event Handling	116
Defining Web Form Control Events	116
Responding to Web Form Events	117
Using the AutoPostBack Property	121
Automatic State Management with Web Forms	123
Summary	126

8 CREATING SIMPLE WEB PAGES WITH THE HTML SERVER CONTROLS 127

What Are HTML Server Controls?	127
The Power of the RunAt="Server" Attribute	128
Why You Can Add RunAt="Server" to Any HTML Tag	129
The HTMLControl Class	130
The General Controls	131
Anchor (<A>)	131
Image (IMG)	132
Form (Form)	133
Division (Div) and Span (Span)	134
The Table Controls	135
Table	135
Table Header (Th), Row (Tr), and Detail (Td)	136
The Input Controls	137
Text, Password, Textarea, and Hidden	138
Submit, Reset, Image, and Button	140
Select	142
File Input	145
Summary	147

9 CREATING INTERACTIVE FORMS WITH WEB FORM SERVER CONTROLS 149

What Are Web Server Controls?	149
The WebControl Class	150
General Controls	151
HyperLink	151
LinkButton	151
Image	153
Label	154
Panel	154
Form Controls	156
TextBox	156

RadioButton	159
CheckBox	162
DropDownList	163
ListBox	166
Button	168
ImageButton	170
Creating Post Away Forms	174
Table Controls	176
Programmatically Generating Tables	177
Summary	180
10 DESIGNING ADVANCED INTERFACES WITH WEB FORM LIST CONTROLS 181	
Using Simple List Controls	182
The RadioButtonList Control	182
The CheckBoxList Control	188
Using Templated List Controls	194
The Repeater Control	195
The DataList Control	198
The GridView Control	203
Summary	214
11 IMPROVING YOUR USER INTERFACES WITH VALIDATION CONTROLS 215	
What Are Validation Controls	215
Properties and Methods Common to All Validation Controls	216
Simple Validation Controls	217
The RequiredFieldValidator Control	217
The CompareValidator Control	219
The RangeValidator Control	220
Advanced Validation Controls	221
The RegularExpressionValidator Control	221
The CustomValidator Control	223
The ValidationSummary Control	229
Summary	230
12 ADDING USER CONTROLS TO YOUR WEB FORMS 231	
What Are User Controls?	231
Simple User Controls	232
Creating a Markup-Only User Control	232
Adding Custom Properties to User Controls	234
Advanced Features of User Controls	237
Handling Events in User Controls	237
Loading User Controls Dynamically	240
Summary	242
PART IV HANDLING DATA ACCESS WITH ADO.NET 243	
13 INTRODUCTION TO ADO.NET AND DATA BINDING 245	
The ADO.NET Dichotomy	246
The Database Classes: Data Providers	247
The Data Classes: The DataSet	247
Data Binding	248
Binding to Properties, Methods, and Functions	248
Binding to Collections	250
Binding Complex List Controls to the DataSet Class	251
Summary	258

14	ACCESSING DATA WITH .NET DATA PROVIDERS	259
	Working with .NET Data Providers	258
	The SQL Server .NET Data Provider	259
	The OLE DB .NET Data Provider	260
	Connecting to Data Using Connections	260
	Executing SQL with Commands	261
	Retrieving a Single Value	265
	Working with Stored Procedure Parameters	267
	Fast Data Access with DataReaders	271
	Reading DataReader Rows	272
	Using a DataReader with a DataGrid Control	274
	Creating DataSets with the DataAdapters	276
	Summary	280
15	WORKING WITH ADO.NET DATASETS	283
	Creating a DataSet	284
	Working with DataTables	284
	Manipulating Rows of Data	286
	Counting Rows	287
	Retrieving Field Values	287
	Filtering, Sorting, and Binding with DataViews	289
	The DefaultView Property	289
	Creating Custom Views	289
	Relating Tables with the DataRelation Object	293
	Fabricating DataSets	295
	Creating Columns	298
	Creating the PrimaryKey Constraint	299
	Adding Rows to a DataSet	299
	Creating a DataSet from XML	300
	Generating XML from a DataSet	306
	Updating DataSet Data	310
	Managing the Changed Status of the DataSet	311
	Writing Updates Back to the Database	312
	Summary	325
PART V	CREATING AND USING WEB SERVICES	327
16	UNDERSTANDING THE WEB SERVICE MODEL	329
	Some Web Service History	329
	The SOAP Protocol	330
	The Web Service Description Language	330
	The SOAP Request Message	332
	The SOAP Response Message	333
	Publishing and Consuming Web Services	334
	Publishing Web Services	334
	Consuming Web Services	335
	Other Important Issues	336
	Security	337
	State Management	338
	Transaction Management	339
	Summary	339
17	PUBLISHING WEB SERVICES WITH ASP.NET	341
	The Basics of Creating Web Services with ASP.NET	341

A Typical ASMX Template	342
Building Public WebMethods	343
Exploring the Default SDLHelpGenerator Pages	344
Adding WebMethod and WebService Attributes	345
Creating Other Web Service Examples	347
Returning ArrayLists	347
Returning Custom Classes	348
Returning Complex DataSets	351
Summary	356
18 CONSUMING WEB SERVICES WITH ASP.NET 357	
Review of WSDL Contracts	357
Services and Bindings	358
PortTypes, Messages, and Schema	360
Creating HTTP Web Service Clients	362
Using HTTP-GET to Retrieve Web Service Data	363
Using HTTP-POST to Retrieve Web Service Data	363
Creating SOAP Web Service Clients	364
Generating a SOAP Proxy with WSDL.EXE	365
Coding an ASP.NET Client Using a SOAP Proxy	367
Summary	369
PART VI CONFIGURING AND DEPLOYING ASP.NET SOLUTIONS 371	
19 TRACING, DEBUGGING, AND OPTIMIZING YOUR ASP.NET APPLICATIONS 373	
Tracing Services for ASP.NET	374
Page-Level Tracing	374
Application-Level Tracing	377
Using the Standalone Debugger	379
A Quick Tour of the Standalone Debugger	379
Debugging ASP.NET Pages	381
Debugging Compiled Components	385
Optimizing Your ASP.NET Applications with Caching Services	386
Implementing Page Output Caching	387
Implementing User Control Output Caching	388
Utilizing Data Caching	390
Summary	392
20 IMPLEMENTING ASP.NET SECURITY 393	
Important Security Concepts	393
Authentication	394
Authorization	394
Impersonation	395
Interaction with IIS	396
Implementing Windows-Based Security	398
Windows-Based Authentication	399
Windows-Based Authorization	401
Implementing Forms-Based Security	402
Forms-Based Authentication	403
Forms-Based Authorization	406
Summary	407
21 CONFIGURING AND DEPLOYING YOUR ASP.NET SOLUTIONS 409	
Configuring Your ASP.NET Solutions	410

Understanding the ASP.NET Configuration Model	410
Configuration Elements	412
Common Configuration Elements in WEB.CONFIG Files	413
Accessing Configuration Data at Runtime	414
Deploying Your ASP.NET Applications	415
The New Deployment Model for ASP.NET Solutions	416
Copying the Application to the Web Server	417
Creating the Virtual Directories on the Web Server	419
Handling Component Registration, Versioning, and Replacement	423
Summary	424

APPENDIX

A MOVING TO VB .NET FROM VB6 OR VBSCRIPT 425

What's New in VB .NET?	425
Moving From VBScript/ASP to VB .NET	425
Moving From VB6 to VB .NET	426
Tightening the Reins	427
VB Does OOP	428
Arrays	429
Try/Catch Error Handling	429
Other Changes	430
Dim Is Brighter	430
New Assignment Operators	430
Where Is...?	431
Features That Have .NET Framework Equivalents but Are Still There	431
Features that Are Gone	432
Summary	432

INDEX 433

About the Authors

Mike Amundsen is an internationally known author and lecturer with more than 15 years of experience in the computer industry. He travels throughout the United States and Europe speaking and teaching on a wide range of topics including .NET, the Internet, team development and leadership, and other subjects. Mike also founded EraServer.NET—a public hosting service that focuses on .NET Webs and XML Web Services.

He has more than a dozen books to his credit. Along with this new book, his most popular titles are *Teach Yourself Database Programming with Visual Basic in 21 Days*, and *Using Visual InterDev*.

When he is not working, Mike spends time with his wife and three children at their home in Kentucky, USA.

Paul Litwin is a programmer, editor, writer, and trainer focusing on ASP, ASP.NET, Visual Basic, SQL Server, XML, Microsoft Access, and related technologies. He is the editor in chief of *asp.netPRO* magazine (www.aspnetpro.com), a magazine for the ASP.NET professional. Paul is the author of a number of books, articles, and training materials on ASP, ASP.NET, Web development, Microsoft Access, and Visual Basic.

Paul is one of the founders of Deep Training, a developer-owned training company providing training on Microsoft.NET (www.deeptraining.com). He is the conference chair of Microsoft ASP.NET Connections (www.asp-connections.com) and speaks regularly at other industry events, including Microsoft TechEd and the Microsoft Office Deployment and Development Conference.

When not programming, writing, editing, or training, Paul enjoys running and spending time with his wife, Suzanne, and two children, Geoff and Anna. You can reach him at paul@litwinconsulting.com.

Robert Lair is president and CEO of Intensity Software Inc., which specializes in building professional, affordable Microsoft .NET solutions. His latest achievements in the software industry include the work he did on IBuySpy, the premiere demo application showcasing the features of ASP.NET. Bob has spoken at a number of developer-oriented conferences, such as ASP Connections and ASP DevCon. Bob has also written numerous technical articles and columns for such magazines as *Visual Studio Programmer's Journal*, *ASP.NET Pro*, and *.NET Developer*. His latest book, *Pure ASP.NET*, was coauthored with Jason Lefebvre and is in stores now.

Jason Lefebvre is cofounder and vice president of Intensity Software Inc., and has been working in the field of software development for more than five years. His latest achievements in the software industry include the work he did on IBuySpy, the premiere demo application showcasing the features of ASP.NET. Jason has also written several articles for *Visual C++ Developer's Journal*. His latest book, *Pure ASP.NET*, coauthored with Robert Lair, is a premium code-intensive reference for the Microsoft .NET Framework.

About the Tech Editor

Ralph Arvesen works for California-based Vertigo Software, creating multi-tier Web applications for a variety of clients. He started his career writing firmware and hardware design for optical inspection systems and began developing professional Windows applications in 1990 using C and Windows SDK. Ralph has used Visual C++, Visual Basic, Visual InterDev, and SQL Server to create a variety of desktop and Web-based applications. He is the coauthor of two books on Visual C++/MFC and has written articles for the publication *Inside Microsoft Visual C++*. Ralph lives in the Texas Hill Country; you can reach him at his Web site: www.springholler.com.

Dedication

This one's for Denise. Through the years, your quiet friendship and steadfast support have given me the confidence and courage to attempt the things you always knew I could accomplish.—MCA

To the newest bundle of joy in my life, Anna Elizabeth. Every day you make me smile and remember to appreciate the wonder of life.—PEL

Acknowledgments

Although the cover of this book bears only two names, without the help of many other people this book would not exist.

First of all, we'd like to thank Bob Lair and Jason Lefebvre. Bob wrote Chapter 6, "Using Namespaces and Assemblies with Visual Basic .NET." And Jason wrote Chapter 11, "Improving Your User Interfaces with Validation Controls." This really helped us when we were in a bind with too many chapters to write and too few hours left in the day. Thanks, guys.

We'd also like to thank our technical editor, Ralph Arvesen, whose careful eye helped catch a few errors that we had missed. It goes without saying that any remaining technical errors are our own fault. In addition, we'd like to thank Ken Getz who helped look over one or more chapters and provide some timely advice.

Once again, the folks at Sams were great. Thank you Neil Rowe, our acquisitions editor. You stuck with us despite our terminal lateness on getting chapters written. We'd also like to thank our esteemed editors, including Elizabeth Finney, Laurie McGuire, Ned Snell, and Audrey Doyle. Thanks for making us sound smarter than we really are.

A big thanks goes to the many wonderful folks on the ASP.NET team at Microsoft, especially Mark Anders, Scott Guthrie, Susan Warren, Rob Howard, Keith Ballinger, and Erik Olson. Extra thanks go out to Susan, who also wrote the all-too-kind foreword to the book.

And last, but hardly least, we'd like to thank our ever-patient families and friends. In particular, Mike would like to thank Lee, Shannon, Jesse and Dana. Paul would like to thank his wonderful wife, Suzanne, and his amazing children, Geoff and Anna. Thanks for cutting us some slack and making things easy on us when we worked way too many hours cranking out chapters. Oh, and thanks for filling the non-writing part of our lives with joy.

Mike Amundsen and Paul Litwin

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an Associate Publisher for Sams, I welcome your comments. You can fax, e-mail, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax: 317-581-4770

E-mail: feedback@samspublishing.com

Mail: Linda Engelman
Associate Publisher
Sams
201 West 103rd Street
Indianapolis, IN 46290 USA

Introduction

Welcome to the world of ASP.NET development! Microsoft's decision to shift the focus of its flagship development tools and technologies from Windows to the .NET Framework means that there are a host of new challenges and opportunities for everyone working to design, develop, deploy, and maintain successful software solutions. Even if you are not currently using Microsoft's Active Server Pages (ASP) to build your Web-based applications, Microsoft's .NET platform is bound to have an influence on how you design and code your applications. So, if you are someone who wants to understand what the .NET Framework and ASP.NET are, how they work, and how they can be used to build winning Web-based applications, this book is for you.

An ASP.NET Quick Start

As the title says, this book is for developers. We assume you are already doing some sort of server-side Web development using Active Server Pages (ASP), Visual Basic 6, Visual InterDev 6, Java Server Pages, or perhaps PHP (Perl Hypertext Processor). So we've dispensed with the basics and have focused on how to get up and running with ASP.NET quickly.

This book is designed to give you a quick start to using ASP.NET. While it covers a wide range of topics and techniques, it does not go into the deepest detail in all areas. The idea is to help you get a grip on the most important issues surrounding the design and implementation of .NET-based applications—not to explore the minute details of them.

While the book is not a comprehensive exploration of ASP.NET, it is a great way for existing programmers to get started. It contains everything you need to get up and running with ASP.NET programming, including an explanation of the platform, a solid introduction to Visual Basic .NET, thorough coverage of the ASP.NET controls and technologies, explanations of ADO.NET and SOAP-based XML Web Services, and configuration and deployment issues.

What's Covered in This Book?

This book covers all the bits you need to learn in order to build successful Web-based apps for Microsoft's .NET platform. To do that, this book is divided into six major parts:

- Understanding the .NET Environment
- Visual Basic .NET QuickStart
- Building Web Pages with ASP.NET
- Handling Data Access with ADO.NET
- Creating and Using Web Services
- Configuring and Deploying ASP.NET Solutions

NOTE

As you undoubtedly know, ASP.NET and the .NET platform support a great variety of programming languages, including C++, C#, Visual Basic, and JScript. To make things simpler for everyone, however, we have decided to use only one language in the text and examples: Visual Basic .NET.

Understanding the .NET Environment

In this part, you learn what the .NET Framework really is, how it works and how you can use it to build reliable, scalable Web-based solutions. You also learn how to design your applications to take advantage of the many .NET features, and get a quick rundown of many of the new and exciting features of ASP.NET.

Visual Basic .NET Quick Start

This part brings you up to speed quickly with this new and powerful version of Microsoft's most popular development language. If you've used Visual Basic before, this section helps you learn the most important changes to the language and gives you a chance to get up and running quickly using the newest release of Visual Basic.

If you've never used Visual Basic before, this part is still very valuable. You'll find clear examples of all the important aspects of writing Visual Basic programs, including variable declarations, basic syntax rules, program control structures, creating and using objects, and how to compile and install components and standalone programs.

Building Web Pages with ASP.NET

In this part, you learn how to use the new ASP.NET coding language to create powerful Web Forms for your apps. You learn how to use the server-side execution environment, take advantage of automatic view-state management, use server-side forms, create your own user controls, and perform validation. Whether you've used ASP before or not, this part gets you up and running fast on the newest version of this powerful technology.

Handling Data Access with ADO.NET

With the release of the .NET platform, Microsoft introduces ADO.NET, the next generation of data access for distributed applications. This part of the book shows you how to take advantage of this new technology to provide faster, more reliable data services to your Web-based solutions. You learn how to connect to databases, retrieve data, update data, and execute stored procedures. You also learn about data binding and the manipulation of disconnected datasets. In addition, you learn how to read and write XML data.

Creating and Using XML Web Services

One of the most intriguing new features of the .NET platform is the ability to create and use SOAP-based XML Web services. In this section of the book, you learn what XML Web services are, how to design and deploy them, and how to create remote Web clients that can use the Internet to access these services.

Configuring and Deploying ASP.NET Solutions

In the final part, you learn how to configure the .NET Framework on your Web servers and how to deploy your solutions to one or more servers. The new .NET Framework makes it easier than ever to quickly deploy and configure the system—especially from a remote location. In this part, you learn how you can perform all these tasks without ever having to register a DLL or stop and start the Web server. You also learn how to implement ASP.NET security, and how to add tracing, debugging, and caching services to your ASP.NET solutions.

What's Not Covered?

This book covers quite a bit of material in a short amount of time. However, there are quite a few details that are beyond of the scope of this book, but still important.

HTML and XML Markup

For example, this book does not spend much time discussing the details of HTML or XML. You'll see lots of complete examples here, but you won't find a thorough explanation of these markup languages and how they can be used to their fullest potential. If you want to learn more about HTML and XML, there are many books on the subject as well as online resources to help you.

Windows Operating Systems and Internet Information Server

This book does not go into great detail on how to use Microsoft's Internet Information Server or Windows operating systems. You'll learn how to make sure the .NET Framework is installed and configured properly to run on the most common Windows OS's and how to use IIS to install and manage your .NET apps. However, if you want to learn more on how to take best advantage of the power of Windows and IIS, you'll need to look elsewhere.

SQL Server

Even though there are many examples of database access in this book, you will not find detailed information on how to install and configure SQL Server or how to use the SQL language to its fullest. This is a huge topic—one that could require several books to cover properly. If you are interested in learning more about SQL Server, or about the SQL query language in general, there are many fine books on the subject.

Visual Studio

This book was developed using a simple text editor, not with Visual Studio. Microsoft's Visual Studio .NET is a powerful development environment, but it is not required to build .NET applications. There will be plenty of other books covering this new and powerful development tool.

System Requirements

There are just a few basic system requirements for you to use the examples in this book.

- A Pentium 233 or higher with 96 MB RAM (or more)
- Windows 2000, Windows XP, or Windows .NET
- SQL Server 7.0, SQL Server 2000, or a later version
- Windows .NET Framework Component Update (or Visual Studio .NET)
- Microsoft Internet Explorer 5.5 (included in the .NET Framework Component and Visual Studio .NET installation) or higher

Most of the examples in this book were developed on one of two machines: a Dell laptop with 128 MB of RAM running Windows 2000 Advanced Server, SQL Server 2000, and the .NET Framework Component Update; or a Toshiba laptop with 256 MB of RAM running Windows 2000 Professional, SQL Server 2000, and the .NET Framework Component Update.

About the .NET Framework, Visual Studio and Beta-Ware

All the material in this book was developed using a beta version of the .NET Framework Component Update. This was a very late beta version of .NET and very little is expected to change once the final product is released. However, it is possible that minor differences in behavior and function will occur. It is important that you test the examples carefully before putting any code into production.

It is important to point out that the code in this book was written without the use of the Visual Studio .NET product. Visual Studio .NET is a powerful development environment, but is not required to create and execute .NET applications. Also, since the Visual Studio .NET developer tool was in early betas at the time this book was written, it would be quite likely that the version we would have shown in this book would have been very different from the one that was available upon final release.

Updates and Changes

As with all “leading-edge” books, this book was written using the most recent beta releases available at the time. Editors and testers have checked the text and the examples to make sure they are in step with the most recent changes in the betas. It’s tough to write these kinds of books, since the very ground upon which they are built is constantly shifting as the product progresses. However, we’ve done the best we can in getting you the most accurate and important information as quickly as possible.

Despite all these efforts, things change and updates are inevitable. To make sure you have the latest information on any changes that relate to this book, you can visit the book’s online Web site at www.aspxdev.net. There you will find any errata, news, notes, and updates to the source code. You’ll also find valuable links to other ASP.NET-related sites that can help you learn more about ASP.NET and the future of Web development using Microsoft tools.

Call to Action

The release of .NET marks a major change in the creation of Web-based applications. This book is designed to put you at the leading edge of that change by helping you understand what .NET really means, how to write applications based on the .NET Framework, and how to configure and deploy servers to take advantage of the power and flexibility of .NET.

The next generation of Web programming is here. Ready? Set? Go!

Mike Amundsen & Paul Litwin

PART I

UNDERSTANDING THE .NET ENVIRONMENT

- 1 Introducing .NET
- 2 Understanding the .NET Framework
- 3 What's New with ASP.NET?

CHAPTER 1

Introducing .NET

The release of the ASP.NET Component Update for the Windows operating system represents what some have called a “geological shift” in the focus and strategy of Microsoft Corporation. Microsoft’s representatives have likened the importance and potential impact of the .NET strategy to the shift Microsoft undertook when it moved the company away from the DOS operating system and on to Windows. More than one pundit has characterized the .NET move as a last ditch attempt to keep Microsoft from losing its longstanding hold on the lion’s share of the software market.

Whether any of these descriptions proves accurate only time will tell. However, there is no doubt that the release of ASP.NET and the associated changes in product focus that Microsoft is undergoing will have a noticeable affect on anyone designing, implementing, and/or supporting Web-based applications today. Without a doubt, anyone currently using Microsoft tools and technologies such as FrontPage, Visual Studio, or just Active Server Pages alone will have to learn to adapt to the major changes introduced with the release of ASP.NET and Visual Studio .NET. But even those not currently using Microsoft development tools and platforms will be affected by the changes being driven by one of the world’s largest software companies.

As is so often the case, much has been written and said about ASP.NET and .NET in general long before most of us have had an opportunity to actually see what they really are. In this chapter, you learn more about what .NET and ASP.NET really mean, how they work, and how the introduction of the .NET platform and ASP.NET will affect the design, implementation, and maintenance of Web-based solutions. In the rest of the chapters in Part 1, “Understanding the .NET Environment,”

you also get details on how the .NET runtime platform works and about what's new in ASP.NET.

The New Face of Microsoft Web Development

ASP.NET represents the “new face” of Microsoft Web development—not just because the development tools are different and not just because the resulting Web applications can be more powerful and flexible than current ASP solutions. ASP.NET also has a number of key features that make it a powerful development platform for building Web applications. Some of these powerful features include a focus on creating distributed applications, support for using the Internet as a key mode of software delivery, and improved operating system and development tools.

The Move from Workstation to Distributed Computing

One of the key differences between ASP.NET and previous versions of Microsoft Web development environments is that the .NET platform upon which ASP.NET is built represents a move from workstation computing to distributed computing. While previous development efforts have assumed the primary execution environment was your local workstation, the .NET platform has been designed to support executing distributed applications. In other words, the .NET platform makes it easier to build solutions that execute remotely.

This can be seen in a number of features in the .NET platform and in ASP.NET in particular. First, ASP.NET now allows you to deploy an application by simply copying the compiled DLLs and the pages to the target server. You no longer need to use registry entries in order to complete an install. You can also easily update an application by just copying over the existing pages and DLLs with new ones. There is no need to restart the IIS Web service or reboot the machine. Finally, the .NET platform upon which ASP.NET is built also provides similar features for desktop applications. All this shows that the .NET platform was designed to support remote installation with minimal impact on the target device—a big step forward for building distributed solutions.

Internet As a Key Mode of Delivery

This focus on distributed solutions also brings to light another key feature of the .NET platform: using the Internet as a key mode of software delivery. Now more than ever, Microsoft needs to be able to deliver its software in the most efficient and friendly way possible. And much of this revolves around taking advantage of the increasingly ubiquitous Internet. For example, the .NET platform has a feature that allows desktop applications to automatically install components via the Internet. This will make updating existing applications rather easy and increase the use of Internet-connected workstations that only download and use features of an application that are necessary.

At the same time, ASP.NET solutions can host XML Web services which allow programmers to create Web components. These expose a familiar type library of methods and properties that can be called from other applications to return simple results instead of Web page user interfaces. This allows programmers to use the Internet as a vast collection of software libraries without even having to download and install components. .NET itself has support for XML Web services built into the very lowest level and this makes accessing XML Web services a relatively simple task using ASP.NET.

Operating System and Development Tools Lead the Way

Finally, Microsoft has focused its efforts at creating this new development environment by creating, first, a new set of operating system components that are at the heart of the .NET platform, and second, a new set of development tools that assist programmers in creating solutions built on the new operating system features. This one-two punch of operating system enhancements and new developer tools is key to Microsoft's plan for creating a successful environment for building the next generation of software solutions.

At the heart of the .NET revolution is the set of operating system components needed to make it all work. Called .NET runtimes, the Windows .NET Component Update, the Common Language Runtime, and so on, this set of low-level components is required for any machine that will host .NET applications. In essence, the .NET runtimes is an execution engine and a vast set of supporting class libraries. These basic parts take up less than 20 MB on the machine and represent the base-level features of .NET.

All .NET applications depend on this set of components in order to run. In fact, all classes, methods, and properties used in pure .NET applications call directly to this component library, and bypass the actual Windows application programming interface (API) completely. This means that the .NET runtimes sit between the program and the operating system providing a generic interface between the two. This is a key feature of the .NET platform (see Figure 1.1).

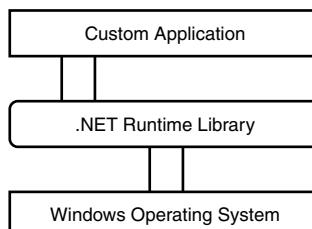


Figure 1.1

The .NET runtimes rest between the application and the operating system.

In order to make it possible to build these new applications, Microsoft has created a new set of compilers and support tools. This set of tools includes command line compilers, debuggers, and profile tools along with a collection of utilities for inspecting the runtime libraries, creating security and access policies, and many others.

NOTE

You can find all the .NET platform utilities in the BIN folder where your version of the .NET framework is installed. The default directory for these files is C:\Program Files\Microsoft .Net\FrameworkSDK\Bin.

One of the more intriguing aspects of this new set of development tools is the existence of command line compilers. For example, with the release of the .NET platform, programmers can now use simple text editors, such as Notepad, to build Visual Basic applications, and compile them using the command line VBC.EXE compiler. No longer is the Visual Basic editor required to build VB .NET applications. This makes it possible for a much wider audience to build and maintain Microsoft solutions at virtually no cost, since the .NET platform runtimes will be released free to the general public.

Of course, in addition to the simple command line tools and text editors, Microsoft is releasing Visual Studio .NET. VS .NET is a full-featured development environment that runs on the Windows operating system. It is also possible to use any number of HTML and code editors to build and compile your .NET solutions.

The Power of the .NET Platform

The real power of ASP.NET lies in the .NET platform itself. This optimized runtime environment represents Microsoft's strongest effort yet to create a programming environment that is at the same time secure, reliable, and scalable. There are a number of powerful features and services within the .NET platform—too many to mention in this book. However, there are four in particular that have a great bearing on the creation of ASP.NET applications that are worth discussing in some detail:

- OS-Neutral Environment
- Device Independent
- Wide Language Support
- Internet-Based Component Services

These four areas all affect the way Web applications in general, and ASP.NET applications specifically, will be designed, deployed, and used in the future.

An OS-Neutral Environment

One of the most important aspects of the .NET Platform is that it is essentially operating-system-neutral. The design of the .NET runtime library makes it possible to build both desktop and Web solutions without ever making direct calls to the underlying operating system at all. This means that ASP.NET developers need to know almost nothing of the Windows operating system in order to build and deploy their solutions.

One of the key advantages of this strategy is the possibility that the .NET platform will be ported to other non-Microsoft operating systems. For example, as of this writing, Microsoft has announced plans to create a version of the .NET runtime on the FreeBSD platform. This could lead to ASP.NET applications running on the Apache Web server.

In addition to the FreeBSD project, a number of other independent groups are working to create .NET runtimes that run on the Linux operating system. Whether these non-Microsoft efforts are fully successful remains to be seen. However, the fact that this is even possible represents a major change in Microsoft's strategy that is worth noting.

Device Independent

Along with the potential for hosting the runtimes on more than one operating system, Microsoft has designed the .NET platform's output to work successfully on a number of different devices. For example, the ASP.NET runtimes contain a set of user interface controls (Web Form controls) that automatically query the calling browser or device and then emit the proper markup language that will work for the calling device. This means that Web application developers no longer are required to write complicated branching code that will send one set of files to Netscape browsers and another set of files to Microsoft browsers. Just one markup page is all that is needed to support multiple browsers.

What is even more interesting is the ASP.NET platform also supports a new set of user interface controls designed to work with mobile devices. The Mobile Internet Tools (MIT) SDK contains a set of controls that can work with common Web browsers, CE devices, Palm devices and a wide range of cell phones. Now, a single markup and code base can be used to display data on a vast range of devices without the need to change the source code or add complicated branching algorithms.

This device independence takes a big leap forward in the ability to create a single source code library that can run successfully on almost any device.

Wide Language Support

Along with the ability to host the platform on various operating systems, Microsoft has also designed the .NET runtimes to support multiple source code languages. This allows programmers to choose the language they are most comfortable with when creating their solutions.

Microsoft .NET Languages

The .NET runtimes ship with support for three Microsoft languages that can be used to build ASP.NET applications: C#, Visual Basic .NET, and Jscript .NET. In addition to these three ASP.NET-compatible languages, Microsoft's Managed C++ is a version of C++ that uses the .NET runtimes and libraries to build desktop applications.

C# is the language Microsoft staff used to build much of the .NET class libraries themselves. It is an extension of the C++ language that has many similarities with Java. Like all the .NET languages, it offers an object-oriented, type-safe, garbage-collected environment that offers the RAD and component feel of Visual Basic.

Visual Basic .NET is a much-updated version of Visual Basic 6.0. Along with the full object-orientation, VB .NET now has all the features and functions of a fully functional Internet programming language. Although some of the changes to VB .NET will make simple upgrading from VB6 to VB .NET less than trivial, VB .NET continues to be a highly effective and efficient language for building solutions.

NOTE

You'll learn more about the changes in VB .NET in Part 2, "Visual Basic .NET QuickStart."

Finally, Jscript .NET is a new version of the ECMA JScript scripting language used in Microsoft's Internet Explorer and as a language for building class ASP solutions. Like the other .NET languages mentioned here, JScript .NET is a fully compiled language that offers great OOP and RAD features. Existing JScript applications can be moved to Jscript .NET with only a minimal amount of changes and can take full advantage of the power and flexibility of the .NET runtime environment.

Other .NET Languages

Microsoft is not the only company working on creating new languages that can be compiled for the .NET runtime system. As of this writing, there are more than 15 .NET languages announced. Many of these are just in the planning stages since it takes quite a bit of time to create compilers and much of the work must wait until the .NET runtime system itself is finally released. However, the growing list of languages shows that several vendors see the .NET runtime system as a viable host for their code base.

Of the many planned languages that are in the works for .NET, at least three have already been released in some beta form. It's also interesting to note that all three of these languages support the ASP.NET runtime environment, too. The three languages available as of this writing are Fujitsu's COBOL.NET, PERL.NET from ActiveState, and Eiffel for .NET from Interactive Software Engineering. All three languages provide access to the base level .NET runtimes and allow programmers to create stand-alone components that can be shared among the other .NET languages. Programmers can also use these languages to build familiar XML Web services, User Controls, ASP.NET Web Forms, and other user interface items.

Internet-Based Component Services

The last feature of the .NET platform that will be covered here is the focus on Internet-based component services. Basically, Microsoft has built into .NET the ability to treat almost any Internet location as a component. The key technology used to accomplish this is XML-based SOAP (Simple Object Access Protocol) Services. SOAP offers a standardized message format for passing information between Web servers over the Internet. Microsoft has used this message system to help create a standardized way to send component and type library information between two machines. This is, in effect, a kind of remote procedure calling (RPC) system that works over standard HTTP connections.

**NOTE**

The details of XML Web services with ASP.NET including the creation and use of ASMX documents are covered in Part 5, "Creating and Using Web Services."

By utilizing the open standards of SOAP, Microsoft has built into the .NET runtime a very simple way to expose components for use across the Internet. In addition, ASP.NET has been designed to make creation of these Internet components rather trivial, too. All you need to do is create a special document in an existing Web (called an ASMX file) and include one or more functions on the page that is marked as Web methods. Once this is done, other Web applications can make calls directly to this ASMX file and, when they pass the proper arguments, can receive the function result as an XML message that can be incorporated into the calling server's pages or other output.

Summary

Microsoft's .NET Platform is an entirely new way to build applications including Web applications. ASP.NET was built on top of the .NET platform and takes full advantage of the power and flexibility of the .NET runtime system including operating system neutrality, device independence, wide language support, and Internet-based component services.

In the next chapter, you'll learn more about the .NET runtime system, the Common Language Runtime (CLR), and how ASP.NET takes advantage of all the features of Microsoft's .NET environment.

CHAPTER 2

Understanding the .NET Framework

At the heart of the .NET Platform is the runtime system itself. This runtime system is called the *Common Language Runtime* (CLR). The CLR is the engine that handles the parsing and execution of all .NET application instructions. CLR also manages all the security and deployment details involved with locating and executing runtime code. The CLR is the foundation upon which the entire .NET platform is built.

On top of this foundation rests a huge library of classes that handle all the mundane work required by programs. These classes are organized into *class libraries* based on the types of tasks they perform. While there are hundreds of classes in the .NET Framework, they can be summed up in a handful of categories:

- **The System Classes**—These handle common low-level tasks required by all the other classes.
- **The Data and XML Classes**—These handle the details of reading and writing data (relational or hierarchical).
- **The Windows and Drawing Classes**—These classes take care of the high-level tasks of drawing items on the screen and managing desktop dialogs and display.
- **The Web Classes**—These classes provide all the support needed to create and manipulate Web pages and other HTTP-related tasks.

While the Windows classes will not be of great interest to Web developers, all the other classes are useful, including the Drawing classes, which allow Web programmers to build solutions that render images and so on. Before getting into the details of the class libraries, it's important to spend a bit of time talking about the CLR itself, how it works, and how ASP.NET uses it.

The Common Language Runtime (CLR)

The Common Language Runtime (CLR) is the runtime system that provides all the power for .NET. It is the foundation of the .NET Platform itself. The CLR provides a number of key services necessary for the support of .NET applications including

- **Code Management**—This includes locating and executing the code, managing the stack and heap, and managing threads, etc.
- **Security Support**—This includes the new evidence-based security model for code access security as well as the higher-level user and group access security system.
- **Garbage Collection and Error Handling**—This includes the optimized GC for .NET as well as the error management and stack tracing and debugging features of the CLR.

These are broad categories and not meant to be totally inclusive of all the features and tasks in the CLR. However, they touch upon the primary activities of the CLR and give you a good idea of the kinds of things going on “under the covers” when you build and execute your ASP.NET applications.

Figure 2.1 shows a diagram that maps out the key internals of the CLR. You’ll notice that all the basic functionality of a complete runtime is encapsulated here. This includes everything from the class loader that handles the loading of your code, through the compiling and execution of the code, on through to the support for the additional class libraries that make up the rest of the .NET Framework.

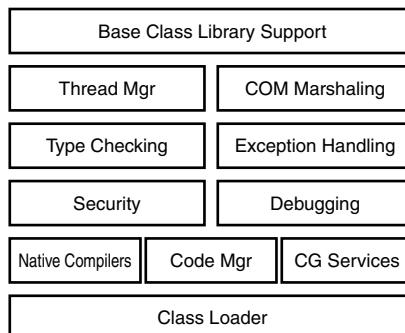


Figure 2.1

CLR internals diagram.

Code Management and Execution

Code management within the CLR covers most of the basic activities of loading and executing .NET programs. One of the key tasks of code management is converting the Intermediate Language (IL) files created by the command line compilers into native code for the target machine. This task is required since the command line compilers do

not create actual executable code. Instead, they create a form of compressed code called IL. This IL must then be converted to native code to run.

Using IL as the primary “payload” for .NET applications has some important advantages. Primarily, this means that “compiling” your code does not lock it into a single operating system or chip set. Theoretically, this means that you can compile your code once and then ship it to many different platforms or devices for execution. That’s because the actual creation of the executable code is handled by the CLR on the host device itself. Figure 2.2 shows how this works.

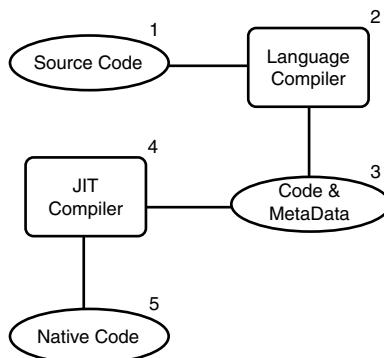


Figure 2.2

CLR compilation and execution cycle.

In Figure 2.2, Step 1 is the creation of the source code. You can use any text editor to create .NET source code files. In Step 2, a command line compiler accepts the source code as input and produces a file (DLL or EXE) that contains parsed code and additional metadata that describes the compiled component (Step 3). In Step 4, the parsed code and metadata is delivered to the just-in-time (JIT) compiler that converts the parsed code into executable code for that operating system and processor (Step 5). For ASP.NET applications, this last step occurs the first time a user requests a page from the Web application.

It is important to point out that this JIT-ed file is stored on disk for future use. Repeated requests to a component will be compiled to native code only the first time. All subsequent requests are routed directly to the native compiled file. This means that all ASP.NET applications execute at “native-code speed” every time. It also means that this conversion to native code happens only once in the lifetime of the component. The only time this changes is when the source code is changed; at that point, the entire loop shown in Figure 2.2 starts again.

Security Support

Along with the support for loading and executing code, the CLR also offers support for a number of security tasks. This includes both code-access security and user- and role-based security tasks.

The CLR has the ability to check a number of security evidence factors. It is the collection of these factors that will determine whether a set of code has the permissions to execute at any given time.

The CLR keeps track of a number of security evidence items. First, the code itself can make permission requests. For example, “Do I have rights to create a new file in this folder?” And the CLR itself can grant these permissions. Permissions can exist for users and roles, too. All these permissions can be tested against predefined permission sets that exist on the machine hosting the code. This allows server managers to control the level of trust granted to code that runs on the machine. Finally, the actual location of the running the code, whether it is being called remotely or locally, as well as the login rights of the user running the code, all play a part in determining the permissions of the executing application.

Figure 2.3 shows how the runtime takes various evidence items and the existing code access policies present on the machine and presents these items to the security policy engine in order to compute the actual permissions of the code.

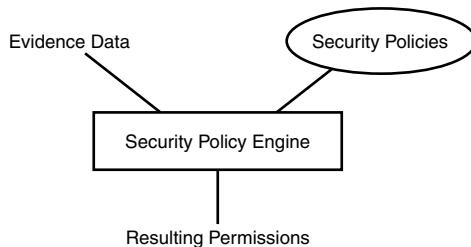


Figure 2.3

Computing code access permissions.

Error Handling and Garbage Collection

Along with code execution and security tasks, the CLR also handles memory management and error handling. The error-handling service within the CLR works on the paradigm of “try and catch.” Any code that might cause an error is wrapped in this try and catch block. If an error occurs in the try block, a selected catch block of code is executed. Usually, the catch block accesses a special exception class that contains details from the CLR about what was attempted and the error that occurred. This is a very robust way to manage errors. It is also a very “portable” error model that works quite well with the various languages hosted on the CLR.

The process of memory management is handled by the optimized garbage collector (GC). The GC keeps track of all the objects and memory variables that have been created and, when these objects are no longer needed, clears them from memory to make room for new variables and objects.

 **NOTE**

The details of memory management with GC services are complex and highly theoretical. For the purposes of this book, it is only important to understand some of the behaviors and implications of the CLR GC instead of the details of how and why it does its magic.

One of the important aspects of the GC in the CLR is that objects are not finalized until the memory is absolutely needed or the executing program ends. This optimization of the process by waiting to recover memory results in much more efficient execution of the programs. However, in some rare cases, finalizing code that is meant to clean up resources handles may not run for some time after the object is no longer in use. This lack of deterministic finalization is an important issue for some programmers. Without going into great detail, it is important to remember to explicitly close any open resource handles before releasing your objects.

The CLR is the low-level runtime that is at the heart of the .NET Platform. However, the CLR alone is not complete enough to host high-level applications such as desktop or Web solutions. To do this, you also need a library of classes that provide the basic services and functions needed to create fully functional applications.

The .NET Framework Class Libraries

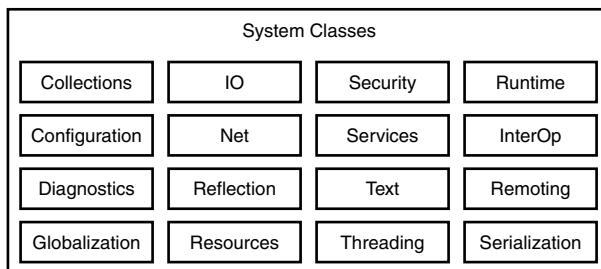
While the CLR is the foundation of the .NET Platform, it's the Framework class libraries (FCL) that offer the key functionality required to build and deploy working ASP.NET applications. The FCL provides hundreds of classes that handle everything from simple math and string operations to the high-level tasks of creating dialogs on the desktop and Web pages in the browser.

Even more important, the FCL is easily expanded and updated by adding new DLLs or updating existing ones. This is an important factor in the design of the FCL. Microsoft has gone to great lengths to make sure it is relatively easy to upgrade and maintain the FCL without having to go through a lot of trouble downloading the entire FCL in order to support a single update or modification. To accomplish this, the FCL is divided into several DLLs, each focusing on a set of related tasks.

The next several sections briefly discuss the main groups of class libraries and the service they offer.

System Classes

The System Classes offer the basic services needed to support all the other classes in the FCL. This includes low-level tasks such as threading, security, remoting, and serialization. It also includes tasks like access to network services, general IO, collections, configuration, globalization, diagnostics, and text handling (see Figure 2.4).

**Figure 2.4**

High-level view of the system classes.

The system classes contain all the functions and services to handle basic math and string operations. They also handle collection management through a large number of classes for handling name/value pairs, lists, and arrays. In addition, there are classes for handling other resources such as images and configuration information.

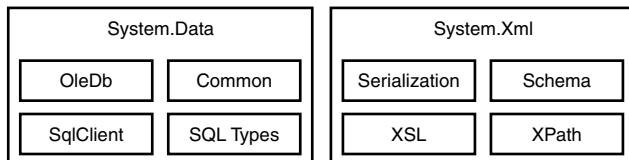
The system classes also handle the task of communicating with CLR for runtime services, handling interop tasks between the CLR and the host operating system, serialization of objects and data, and the process of remoting information across processes and servers. The task of providing reflection (a way for classes to inspect each other's content at runtime) is also handled by the system classes.

Finally, general resource tasks such as globalization, diagnostics, configuration and basic input/output (IO), and network services are handled in the system classes.

Data and XML Classes

Along with the basic classes, the FCL also provides classes for handling data and XML. The System.Data classes include support for OLEDB connected databases (OleDb) as well as an optimized set of classes for accessing Microsoft SQL Server directly (SqlClient and SqlDbType). These data classes comprise what Microsoft calls ADO.NET.

In addition to the data classes, the System.Xml classes offer support for Schema, XPath, XSL, and general serialization of XML data. Finally, all the data services share a common class set for handling basic tasks (see Figure 2.5).

**Figure 2.5**

Data and XML classes.

Windows Forms and Drawing Classes

Another set of classes offer access to basic desktop services such as drawing, printing, dialog design, and user interface component support. This is the work of the Windows Forms and Drawing classes (see Figure 2.6).

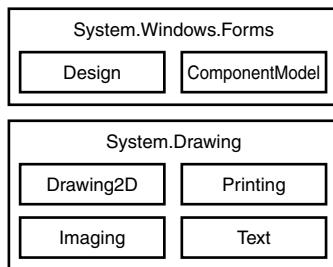


Figure 2.6

Drawing and Windows Forms classes.

While the Windows Forms classes are of little interest to ASP.NET developers, the Drawing classes are quite useful. These classes can be used to perform image-creation and manipulation and generation of complex fonts and other drawing services. The Drawing classes also offer access to printing and other text handling tasks that can be handy for rich client applications.

Web Classes

Finally, the Web Classes are the ones that handle the user interface and Web Services tasks for Web applications. This is the portion of the FCL that ASP.NET users will most often access (see Figure 2.7).

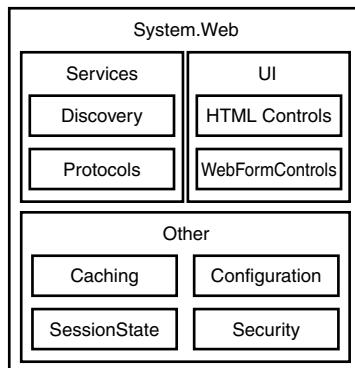


Figure 2.7

System Web classes.

There is a set of basic UI classes that provide support for Web pages, HTML controls, and the new Web Forms controls. These classes contain all the standard HTML controls along with a set of “smart” controls such as grids, data lists, ad rotation, calendar controls, and other high-level display services.

There is also a set of classes that provide access to SOAP-based XML Web Services such as the SOAP protocols, dynamic discovery of Web services, the handling of SOAP proxy calls, and so forth. These form the heart of the XML Web Services model for ASP.NET solutions.

Finally, there are classes to handle other typical Web application tasks such as page and output caching, application configuration, session management, and Web-based security services.

Summary

The .NET Framework is the heart and soul of the Microsoft .NET Platform. And the foundation of .NET is the Common Language Runtime (CLR). It is the CLR that performs the tasks of compiling the source code, managing basic security, garbage collection, general memory management, and other base-level services.

Along with the CLR, the Framework Class Libraries (FCL) provides the functions and services needed to build a fully functional application. The FCL contains hundreds of classes that can be collected together in related groups.

- **System** classes handle the basic math and string operations as well as general IO, network communication, and other services.
- **Data** and **XML** classes provide access to relational and hierarchical data including OLEDB and SQL Server databases along with XPath, XSL, and other XML-related services.
- **Windows Forms** and **Drawing** classes supply printing, drawing, and image-handling services as well as access to desktop dialogs and user interface components and controls.
- **Web** classes provide access to the HTML and Web Forms controls for creating ASP.NET pages, access to the XML-based SOAP Web Services, and other tasks such as caching, session state, configuration, and security services.

Now that you have a basic understanding of .NET itself and the CLR and Framework classes that make up the .NET platform, you’re ready to dive into the specific features of ASP.NET that are available for creating robust and reliable Web solutions.

CHAPTER 3

What's New with ASP.NET?

The previous two chapters presented some important principles—the basic theories and goals of Microsoft’s new .NET initiative and details of the software foundation upon which .NET is based. In this chapter, you’ll get a quick overview of ASP.NET itself.

You’ll learn how ASP.NET offers a new way to build Web applications with “smart” Web Forms and powerful Web Services. You’ll also learn some of the infrastructure features of ASP.NET, including a powerful security model, improved deployment and update features, easier configuration, and increased scalability and availability for your ASP.NET solutions.

ASP.NET Web Forms and Web Services

ASP.NET offers a new, evolutionary approach to building Web applications on the Microsoft platform. This includes an improved, object-oriented programming model for creating Web pages, built-in support for multiple client browsers and devices, and clear separation between markup and execution code.

Improved Web Page Authoring

In the past, a Microsoft Web author was limited to standard HTML markup and client- and server-side interpreted scripting. This meant that both code and markup often were mixed on the same page, making it difficult to update and maintain solutions over time. Also, programmers had to constantly switch between server-side and client-side coding in order to create fully functional solutions. However, ASP.NET introduces solutions for these problems.

Simplified Programming Model

ASP.NET supports a complete language model including Visual Basic, C#, JScript.NET, and many other third-party languages. In addition, ASP.NET solutions are fully compiled on the server. The results are increased performance and improved reliability. Even though all Web pages are compiled, ASP.NET programmers can still use the familiar text-based coding model by simply posting ASCII text pages to the Web server and allowing the ASP.NET runtime to perform the compilation automatically, as needed.

Finally, the programming paradigm for ASP.NET Web Forms is the familiar event-driven model. This is the same paradigm used by rich-client desktop applications such as the one used to build Windows forms applications. Consequently, it is now much easier for Windows programmers to build Web applications without having to learn a whole new set of programming rules and processes.

Multiple Client Support

The ASP.NET Web Forms model also provides automatic support for multiple client browsers and devices.

In the past, Web programmers had to create pages that searched for the browser type and version that was calling the page and then include customized code that took advantage of the features of each of the many browser types currently available. Now, ASP.NET Web Forms can automatically query the browser type and then determine the best mix of server and client code to use to support the calling device.

In some cases, this means that the ASP.NET runtime will send a great deal of client-side DHTML to the calling client, to limit the number of server-side postbacks and create a rich experience for the user. In other cases, the exact same ASP.NET Web Forms will send simple HTML to the browser in order to make sure that the page will work properly.

Separating Code from Content

Another key feature of ASP.NET Web Forms is their ability to easily separate execution code from page markup. In the past, it was almost inevitable that you would need to mix HTML markup and ASP scripting in the same page. As a result, rendering the markup without first completing all the execution code was almost impossible. Updating and maintaining ASP pages also was complicated since even simple changes in execution code could break HTML markup as well.

By default, ASP.NET Web Forms now offer a clear separation between code and markup. First, by offering a fully event-driven, object-oriented programming model, Web Forms simplify the creation of source code documents that separate code and markup. Also, ASP.NET supports *code-behind* format for Web pages. Code-behind capability allows you to place all execution code in an entirely different physical file. Listings 3.1 and 3.2 show an example of the code-behind format.

Listing 3.1 CODEBEHIND.ASPX—ASP.NET Web Form with HTML Markup and No Execution Code

```
%@ Page Description="Code-Behind style" Inherits="CBPage" Src="CBPage.vb" %>
<html>
<body>
<h2 id="header" runat="server" />
<hr />

<form runat="server">
<asp:TextBox id="nameInput" runat="server"/>
<asp:Button id="submit" OnClick="submit_click" runat="server"/>
</form>
<asp:Label id="postBack" runat="server" />
</body>
</html>
```

Listing 3.2 CBPAGE.VB—An Example of ‘Code-Behind’ for a Markup Page

```
Option Strict Off
```

```
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls
```

```
Public Class CBPage : Inherits Page
```

```
    Public nameInput As TextBox
    Public header As HtmlGenericControl
    Public submit As Button
    Public postBack as Label
```

```
sub Page_Load(sender as Object, e As EventArgs)
```

```
    if Page.IsPostBack=false then
        submit.Text="Click me"
        header.innerHTML="In-Page Coding Style Web Form"
    end if
```

```
end sub
```

```
sub submit_click(sender as object, e as EventArgs)
```

```
    postBack.Text="Welcome back, " & nameInput.Text
```

```
end sub
```

```
end Class
```

You can see in Listing 3.1 the page declaration has the `Inherits` and `Src` attributes that link the execution code file (CBPAGE.VB) to the markup page (CODEBEHIND.ASPX). Modifying either the code or the markup without breaking the page is now much safer and easier.

ASP.NET Server Controls

Much of the power of the new ASP.NET Web Forms model comes from the use of the new server-side controls, or *Web Controls*. Web Controls offer a number of important features that make building ASP.NET Web Forms much easier. For example, Web Controls allow programmers to build Web Forms using XML-type markup in the page that ASP.NET will interpret at runtime in order to generate customized code based on the device calling to request the document.

Also, Web Controls automatically support custom rendering, view state, and postback. This makes the controls much more powerful and easier to work with. Finally, ASP.NET supports a number of high-level '*templated*' controls. These controls allow programmers to use XML-like markup to determine the look and feel of the control at runtime, instead of locking down look and feel at design time.

Declarative Programming

Another instance of the power of ASP.NET Web Forms, *declarative programming*, handles much of the work that used to require extensive scripting. For example, in the past Web programmers needed to create libraries of client- and server-side script that could perform user-input validation at runtime. If the solution needed to support a wide range of browser types and versions, this could be a difficult scripting task.

Now, with ASP.NET Web Forms, programmers can use a simple XML-markup tag called a *validation control* that handles all aspects of input validation without adding any client- or server-side code at design time. Listing 3.3 shows an example of using an input validation control in an ASP.NET Web Form.

Listing 3.3 VALIDATE.ASPX—An Example of Declarative Input Validation

```
<%@ page description="validation example" %>
<html>
<body>
<h2>Input Validation Example</h2>
<hr />

<form runat="server">
Enter a string:
<asp:Textbox id="inputBox" runat="server" />
<asp:RequiredFieldValidator id="inputBoxVal"
controltovalidate="inputBox" runat="server">
Error: This is a required field</asp:RequiredFieldValidator>
<br />
<asp:Button id="submit" Text="Submit" runat="server" />
</form>
</body>
</html>
```

Figure 3.1 shows the error display when users attempt to press the submit button without entering a value in the input box.

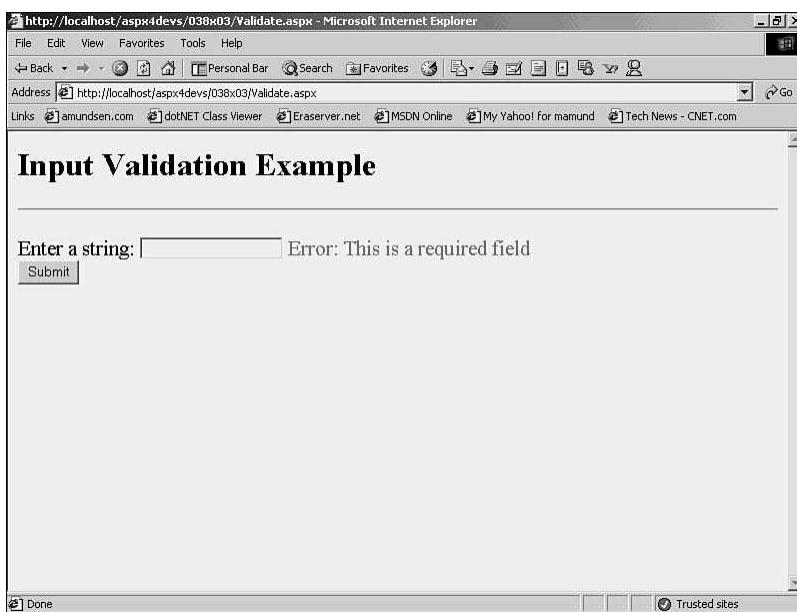


Figure 3.1

The results of posting an empty field in an input validation form.

Support for ViewState and Postback

The ability of server controls to handle viewstate management and automatic postback services without any programmer intervention is yet another example of ASP.NET's power. In classic ASP coding, programmers needed to handle issues such as complex rendering, restoring input values on refresh, and posting the input back upon the same page to perform validation. This added a great deal of work and potential for errors when building user-friendly input forms.

In contrast, ASP.NET Web Forms builds all this intelligence into the server controls themselves. Programmers no longer need to know the details of the HTTP request headers in order to build easy-to-use input forms. Listing 3.4 shows an example that takes advantage of the viewstate and postback features of ASP.NET Web Forms.

Listing 3.4 POSTBACK.ASPX—An Example That Shows postback and viewstate Features of ASP.NET

```
<%@ page description="postback example" %>
<script language="vb" runat="server">
sub myButton_click(sender as object, args as EventArgs)
myLabel.Text=myTextBox.Text
end sub
```

Listing 3.4 continued

```
</script>
<html>
<body>
<h2>PostBack Example</h2>
<hr />

<form runat="server">
Enter a string: <asp:TextBox id="myTextBox" runat="server" />
<asp:Button id="myButton" Text="Submit"
OnClick="myButton_Click" runat="server" />
</form>
You entered: <asp:Label id="myLabel" runat="server" />
</body>
</html>
```

You should notice that the code block in Listing 3.4 does not address the `HTTP request` object at all. That is because the state of the input control is maintained after posting back to the server. Also notice that the `HTML form` control does not contain an `action` or `method` attribute. This is also handled by the `ASP.NET Web Forms`. Finally, after entering a string and pressing the submit button, the contents of the input box are retained (see Figure 3.2).

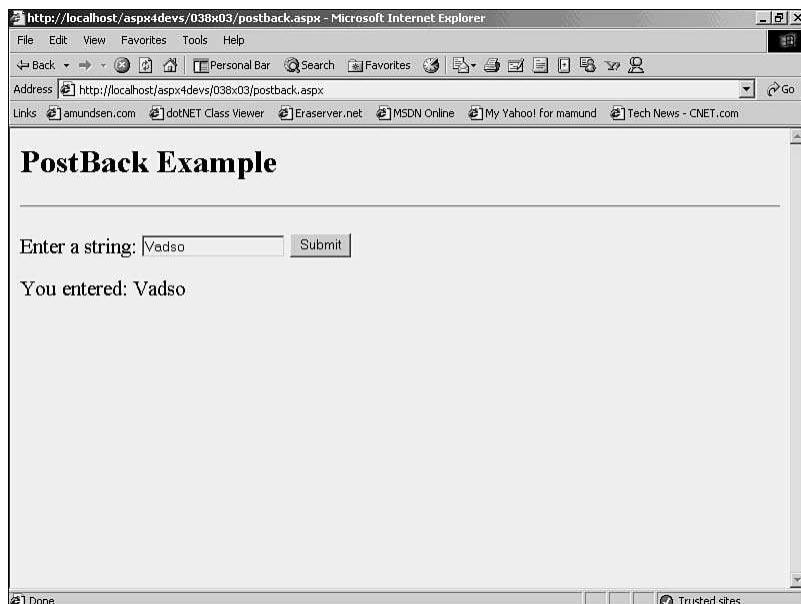


Figure 3.2

Running the POSTBACK.ASPX page in a browser.

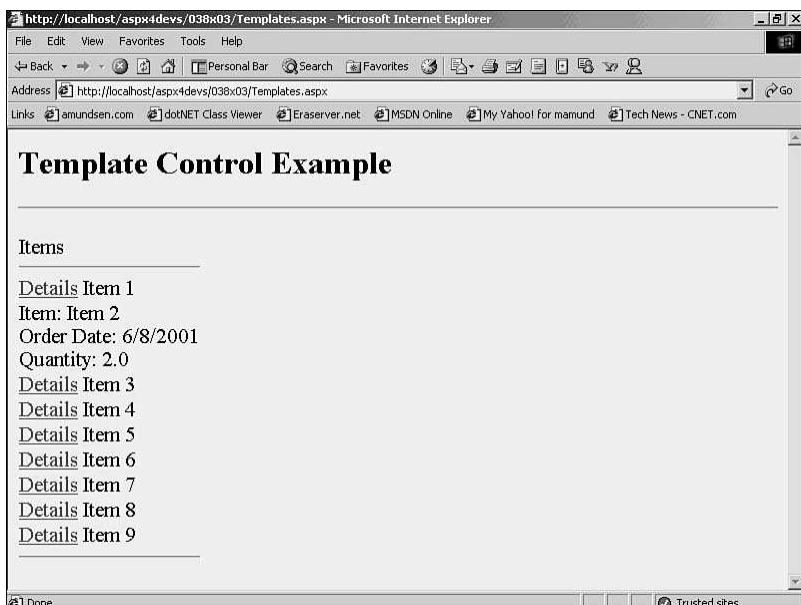
Templated Controls

Finally, ASP.NET Web Forms offers a set of high-level controls that allow you to define your own look and feel for the display. You use a set of templates that define the display at design time, and let ASP.NET render the display at runtime. Listing 3.5 shows an example of a templated `DataList` control in an ASP.NET Web Form.

Listing 3.5 TEMPLATES.ASPX—An Example of the Templat ed DataList Control

```
<form runat=server>
<asp:DataList id="DataList1" runat="server"
OnItemCommand="DataList_ItemCommand">
<HeaderTemplate>
Items
<hr />
</HeaderTemplate>
<ItemTemplate>
<asp:LinkButton id="button1" runat="server"
Text="Details" CommandName="select" />
<%# DataBinder.Eval(Container.DataItem, "StringValue") %>
</ItemTemplate>
<SelectedItemTemplate>
Item:
<%# Container.DataItem("StringValue") %><br>
Order Date:
<%# DataBinder.Eval(Container.DataItem, "DateTimeValue", "{0:d}") %><br>
Quantity:
<%# DataBinder.Eval(Container.DataItem, "IntegerValue", "{0:N1}") %><br>
</SelectedItemTemplate>
<FooterTemplate>
<hr />
</FooterTemplate>
</asp:DataList>
</form>
```

You can see in Listing 3.5 that the `DataList` control has four different templates defined. One template defines the display of the list header and one defines the list footer. There is also a template that defines the display of each item in the list as well as a special template to define the item that has been selected by the user. Figure 3.3 shows how this looks when loaded in a browser.

**Figure 3.3**

Testing the templated `DataSet` control.

ASP.NET Web Services

Possibly the most innovative aspect of ASP.NET is the ability to build SOAP-based (Simple Object Access Protocol) Web remote procedure invocations called *Web Services*. The Web Service feature of ASP.NET lets programmers publish components via the Internet, enabling others to call into those components directly using the SOAP open standard for communicating between machines. This feature brings a powerful distributed computing model to Web applications.

Easy Programming Model

While the SOAP interface is powerful, it also involves a complex mix of XML, protocol stacks, and message handling at the sockets level. However, ASP.NET allows programmers to create these powerful components without having to deal with XML or sockets at all. All you need to do is build a special ASMX text template that contains a class and set of methods that you wish to expose to the Internet and then post this document in your ASP.NET Web.

Listing 3.6 shows an example of a simple Web Service built using ASP.NET.

Listing 3.6 WEBSERVICE.ASMX—A Simple ASP.NET Web Service

```
<%@ webservice class="wsBasic" language="vb" %>
imports System
imports System.Web.Services
```

Listing 3.6 continued

```
public class wsBasic : Inherits WebService
<WebMethod()>public function DoMath(ByVal Value1 as double, _
ByVal Value2 as double) as double
return(Value1*value2)
end function
end class
```

After placing this ASMX document in a public Web, users can execute the DoMath method with nothing more than a simple anchor tag in an HTML document:

```
<a href="wsbasic.asmx/DoMath?value1=12&value2=12>DoMath</a>
```

Support for Multiple Protocols

ASP.NET Web Services also support multiple protocols when communicating with the remote server that is publishing Web methods. For example, ASP.NET supports HTTP-get (anchor tags), HTTP-post (HTML forms), and SOAP (XML messaging). In addition, ASP.NET allows programmers to define their own custom protocol format. This capability makes it possible for ASP.NET to support new, more powerful communications protocols as they become available in the future.

ASP.NET Infrastructure

Along with an improved programming experience, ASP.NET also offers a number of infrastructure innovations that make ASP.NET more secure, easier to configure and deploy, and more scalable and reliable.

The following sections summarize just a few of the more important improvements in the ASP.NET infrastructure.

Powerful Security Model

ASP.NET now supports three different authentication schemes—Windows, Forms-Based, and Passport authentication. This means that you can integrate existing powerful security models into your application with a minimum of coding. Even better, in most cases, you can control the scheme using just a configuration setting, without having to change your code at all.

In addition to the authentication systems, ASP.NET also supports both URL- and access control list- (ACL-) based authorization services. This allows programmers to determine if authenticated users have the proper rights to perform the requested task or view the requested document. The built-in authorization scheme supports both user- and role-based security, too.

Improved Deployment and Update

In the past, deploying and updating most classic ASP solutions could involve gaining access to the registry on the remote server, stopping the Internet Information Server service, and possibly even rebooting the server itself. Now, with ASP.NET, deployment and update is extremely easy and reliable.

Deploying an ASP.NET solution requires only the ability to copy both pages and compiled components to a target Web root. This can be done with a simple DOS `xcopy` command, an FTP script, or even just dragging and dropping files from one machine to another.

Even better, there is no need to make entries in the Web server's registry and no need to run `regsvr.exe` against any compiled components. Instead, updating compiled components on a Web server is as simple as copying the new DLL over the old one. There is no need to restart Internet Information Server or reboot the server. As soon as the new DLL appears on the server, it is available to users of the public Web.

Easier Configuration

One of the main reasons that deployment and update of ASP.NET solutions is so easy is that most of the important information about the Web application is now kept in an XML-based configuration file. In fact, the ASP.NET runtime itself uses an XML configuration file to control most of its runtime behavior. As a result, programmers and Web administrators can quickly and easily update the settings of both Web servers and Web applications using simple text-based tools instead of special registry editors and metabase tools.

Listing 3.7 shows an example configuration file.

Listing 3.7 SAMPLE_WEB.CONFIG—A Sample Web Configuration File

```
<?xml version="1.0" encoding="UTF-8" ?>
<configuration>
<appSettings>
<add key="PubsDSN"
     value="server=myServer;database=pubs;userid=sa;password=' ' " />
<add key="dialogLevel" value="expertMode" />
<add key="defaultBackgroundColor" value="white" />
<add key="defaultFontColor" value="black" />
</appSettings>
<system.web>
<customErrors mode="On" defaultRedirect="ErrorGeneric.aspx">
<error statusCode="404" redirect="err404.aspx"/>
</customErrors>
<sessionState
mode="inproc"
stateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1;user id=sa;password="
cookieless="false"
timeout="20"
/>
</system.web>
</configuration>
```

Increased Scalability and Availability

Lastly, ASP.NET has a number of improvements that help increase overall scalability and availability of your Web solutions, including improvements in session state, new data caching options, and the power to automatically restart services when memory gets low.

Improved Session State

ASP.NET allows you to configure session state management to use one of three supplied providers. You can use the default session state manager that saves all state information in memory on the local machine. This is the same scheme used in classic ASP.

However, you can also use the new “shared session state” manager. This allows you to indicate a server in a Web farm that will hold all the session state data for the entire Web farm. Now session state can be used in a load-balanced server cluster without any trouble at all. Finally, you can use the SQL state manager that allows you to store all session data in a special SQL Server database.

Most important, all these settings are transparent to your programs. You do all this work in the configuration files, not in your code. This means you can change the session-management scheme at any time in the life of your application without needing to change code in any way.

Data and Output Caching

ASP.NET also provides two different caching systems you can use in your ASP.NET solutions. First, you can use output caching for pages that you wish to keep in memory for a fixed time frame. All you need to do is add an output caching directive to the pages you wish to keep in memory. For example, the following code snippet will instruct ASP.NET to keep the page in memory for one hour.

```
<%@ OutputCache Duration="600" %>
```

When ASP.NET loads the page for the first time, it will keep the page in memory, and all subsequent requests for that page will be served from memory instead of executing the code.

You can also employ a new data caching service for your solutions. This allows you to place any object or collection into a special memory location and then recall it at any time. The data caching service is especially handy for database lists that do not change very often but are displayed frequently in an application such as a list of menu options.

Automatic Restarts

ASP.NET supports settings that control just how the runtime will behave in stressful situations, such as low memory or a high number of page requests.

This is handled using the `<processmodel>` element of the configuration files. For example, the following code snippet shows how to control the runtime behavior of ASP.NET.

```
<processModel enable="true"  
requestLimit="200000"  
requestQueueLimit="500"  
memoryLimit="40"  
/>
```

This entry tells the runtime that it should restart processes every 200,000 page requests, whenever the page queue grows past 500, or when the available free memory falls below 40MB.

The capacity for automatic restarts makes it much easier to build robust Web solutions that will perform a level of “self-healing” in stressful situations.

Summary

ASP.NET offers a number of important improvements over classic ASP. These improvements include programming changes, such as new, powerful Web Forms and Web Services that offer event-driven object-oriented programming, declarative smart controls, and separation between code and markup. In addition, improvements to the ASP.NET runtime infrastructure are powerful security services, improved deployment and updated features, easier configuration, and increased availability and reliability.

PART II

VISUAL BASIC .NET QUICKSTART

- 4 Understanding Visual Basic .NET Syntax and Structure
- 5 Working with Numbers, Strings, Dates, and Arrays in Visual Basic .NET
- 6 Using Namespaces and Assemblies with Visual Basic .NET

CHAPTER 4

Understanding Visual Basic .NET Syntax and Structure

All the examples in this book are written in Visual Basic .NET. Why, you may ask, have we decided to use Visual Basic exclusively since the .NET platform supports a plethora of languages? Why not pepper the text with examples in C#, JScript, and maybe even Eiffel?

We decided to concentrate our efforts on only one language to simplify things and to keep the book to a reasonable length. While it's certainly nice to be able to develop ASP.NET applications using a number of different languages, let's face it—most programmers prefer to program in a single language.

But why have we decided to use Visual Basic? After all, isn't C# now Microsoft's *preferred* language? Quite the contrary: Visual Basic is now on equal footing with C++ and the new C#. In addition, we have chosen to use Visual Basic .NET in this book for several reasons: Visual Basic is the most popular programming language in the world. It's also by far the most common language that existing ASP developers have used to create "classic" ASP pages. Finally, it's the language that the authors of this book cut their teeth on—the language that we personally prefer to use.

More than likely, you fall into one of three categories of Visual Basic (VB) developers:

1. You have little or no experience developing applications with Visual Basic or the VBScript scripting language.
2. You have considerable experience developing ASP applications using VBScript but little or no experience with VB proper.
3. You have considerable experience using the Visual Basic language (and perhaps VBScript as well).

This chapter introduces the Visual Basic .NET language to you, regardless of which of these three groups you fall into. For VB novices, this chapter will bring you up to speed in a hurry. For VBScripters, this chapter will help you make the jump from VBScript to Visual Basic. And finally, for the savvy VB developer, this chapter will help you scope out the changes made to your trusty old language.

NOTE

This chapter and the other chapters in this book discuss and use the Visual Basic .NET language, but not the Visual Basic .NET product that's part of Visual Studio.NET. You do not have to own Visual Studio .NET to use the examples in this book.

The New Look of Visual Basic

To borrow the catch phrase of a now defunct U.S. car manufacturer, “This is not your father’s Visual Basic!” While true to its heritage, Visual Basic .NET is a much-improved version of the venerable Visual Basic language that many of us have grown to love. Visual Basic has matured into a full-featured, object-oriented language. But unlike previous releases of Visual Basic, Visual Basic .NET was rebuilt from the ground up. Literally.

In moving to Visual Basic .NET (VB .NET), Microsoft ditched a number of older, arcane features like `GoSub` and default properties, and totally reworked features like arrays and data types. Other native features, like the `MsgBox` function, have been deprecated. These deprecated features are still there in VB .NET, but Microsoft recommends that you move to using the .NET System classes instead. Of course, depending on your experience and base of existing, legacy VB applications, some of the changes may cause you considerable pain. More than likely, however, you will soon grow to appreciate the redesigned VB language.

What does the new Visual Basic .NET language mean to the average ASP developer who has written thousands of lines of VBScript code but who has had little exposure to VB proper? If you find yourself in this category of developer, you may experience a short period of bewilderment, as you get accustomed to the wealth of new features offered by VB .NET—features that VBScript never offered. But soon enough, you will start to forget the limited VBScript language and grow to appreciate and even love the much more nimble and full-featured VB .NET.

NOTE

See Appendix A, “Moving to VB .NET from VB6 or VBScript,” for more information on upgrading to VB .NET.

Getting Started with Visual Basic .NET

Compared to many programming languages, Visual Basic .NET is a fairly easy language to learn. Unlike the C family of languages, VB .NET prefers to use the English language rather than cryptic symbols like `&&`, `||`, and `%`. Unlike prior versions of the VB language, however, VB .NET is a full-featured object oriented language that can hold its own when compared to C++, C#, or Java.

The remainder of this chapter consists of a walkthrough of the essential elements of the VB .NET language.

Statements and Lines

VB .NET statements can be placed on one or more lines. Unlike C++, C#, and Java, there is no statement terminator character in VB. When continuing a statement across more than one line, you must end continuation lines with a space followed by an underscore character (`_`).

For example, the following VB .NET statement spans two lines:

```
Function CreateFullName(LastName As String ,  
    FirstName As String)
```

Comments

You can add comments to your code using the apostrophe (') character. Everything to the right of an apostrophe is ignored by the VB .NET compiler, as shown in the following snippet.

```
x = y + 5 'Add 5 to the value of y
```

NOTE

VB .NET does not support multi-line comments like some other languages.

Operators

Like any programming language, VB .NET has its assortment of operators. The most common of these operators are summarized in Table 4.1.

Table 4.1 Common VB .NET Operators

Type	Operator	Purpose	Example
Math	+	Add	$5 + 2 = 7$
	-	Subtract	$5 - 2 = 3$
	*	Multiply	$5 * 2 = 10$
	/	Divide	$5 / 3 = 2.5$
	\	Integer Divide	$5 \backslash 2 = 2$
	^	Exponentiation	$5^2 = 25$
	Mod	Remainder after integer division	$5 \bmod 2 = 1$
String	+	Concatenate	"one" + "two" = "onetwo"
	&	Concatenate	"one" & "two" = "onetwo"
Assignment	=	Assigns the value of an expression to the variable	$x = 5 + 3$
	+=	Adds the value of a variable to an expression and assigns the result to the variable*	$x += y$
	-=	Subtracts the value of a variable from an expression and assigns the result to the variable*	$x -= y$
	=	Multiplies the value of a variable by an expression and assigns the result to the variable	$x *= y$
	/=	Divides the value of a variable by an expression and assigns the result to the variable*	$x /= y$
	\=	Integer divides the value of a variable by an expression and assigns the result to the variable*	$x \backslash= y$
	&=	Concatenates the value of a variable to an expression and assigns the result to the variable*	$x &= y$
	^=	Exponentiates the value of a variable by an expression and assigns the result to the variable*	$x ^= y$
Comparison	=	is equal to	If ($x = y$)
	<	is less than	If ($x < y$)
	<=	is less than or equal to	If ($x <= y$)
	>	is greater than	If ($x > y$)
	>=	is greater than or equal to	If ($x >= y$)
	<>	is not equal to	If ($x <> y$)
	Like	Matches a pattern*	If ($x \text{ Like } "p??r"$)

Table 4.1 *continued*

Type	Operator	Purpose	Example
Comparison	Is	True if both object variables refer to same object	If (x Is y)
Logical	And	True if both expressions are true	If (x = 3 And y = 4)
	Or	True if one or both expressions are true	If (x = 3 Or y = 4)
	Not	True if the expression is False	If Not (x = 5)
	Xor	True if one expression is true, but not both*	If (x = 3 Xor y = 4)

* This operator was introduced in VB .NET.

You will find a number of examples that use the VB .NET operators scattered throughout this chapter.

Using Procedures

The basic unit of executable code in VB .NET, as in most programming languages, is the *procedure*. VB supports two basic types of procedures: the subroutine (or sub) and the function.

Subroutines

You declare a subroutine with the Sub statement. For example:

```
Sub HelloWorld()
    Response.Write("Hello World")
End Sub
```

You call a sub using either of the following statements:

```
HelloWorld()
Call HelloWorld()
```

NOTE

Unlike prior versions of VB, VB .NET requires parentheses around argument lists whether or not you use the Call keyword.

Functions

Functions in VB .NET are similar in functionality to subroutines with one difference: Functions can return a value to the calling program. You create a function with the Function statement. For example, the following function returns “Hello World” to the calling code:

```
Function SayHello()
    Return "Hello World"
End Function
```

NOTE

Prior versions of VB used a different syntax for returning a value from a function.

Using Variables and Parameters

You use the `Dim`, `Private`, `Protected`, `Friend`, or `Public` statement in VB .NET to declare a variable and its data type. Which statement you use depends on where you wish to declare the variable.

To declare a variable from within a subroutine or function, you use the `Dim` statement. For example:

```
Function DoSomething()
    Dim Counter As Integer
End Function
```

A variable declared using `Dim` is local to the procedure in which it is declared.

To declare a variable that's global to the entire page, you declare the variable outside of any subroutine or function using the `Private` statement. (For backwards compatibility, `Dim` also works in this context, but it's best to use `Private` instead.)

NOTE

The `Public`, `Friend`, and `Protected` statements are discussed later in the chapter, in the section titled "Accessibility of Inherited Properties and Methods."

With a new feature of VB .NET, you can both declare a variable and set its initial value in one statement. For example:

```
Dim Age As Integer = 23
Private Company As String = "Microsoft"
```

VB .NET supports the data types shown in Table 4.2.

You may have noticed in Table 4.2 that there is no entry for `Variant`. That's because VB .NET no longer supports the `Variant` data type. However, you can use the generic `Object` type any place you would have used `Variant` in prior versions of VB. (In VB .NET, "Variant" is a synonym for "Object.")

If you use a declare statement like the following in VB .NET, all three variables will be declared as integers:

```
Dim x, y, z As Integer
```

In prior versions of VB, `x` and `y` would be declared as variant variables, and only `z` would be declared as an integer.

Table 4.2 Visual Basic .NET Data Types

Visual Basic Data Type	.NET Runtime Data Type	Storage Size	Range of Values	Default Value
Boolean	System.Boolean	4 bytes	True or False	False
Byte	System.Byte	1 byte	0 to 255 (unsigned)	0
Char	System.Char	2 bytes	1 Unicode character	“ ”
Date	System.DateTime	8 bytes	January 1, 0001 to December 31, 9999 12:00:00 AM	January 1, 0001 12:00:00 AM
Decimal	System.Decimal	12 bytes	+/-79,228,162,514,264,337,593,543, 950,335 with no decimal point; +/-7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest non-zero number is +/-0.00000000000000000000000000000001	0.0
Double (double-precision floating-point)	System.Double	8 bytes	-1.79769313486231E308 to -4.94065645841247E-324 for negative values; 4.94065645841247E-324 to 1.79769313486232E308 for positive values	0.0
Integer	System.Int32	4 bytes	-2,147,483,648 to 2,147,483,647	0
Long (long integer)	System.Int64	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
Object	System.Object	4 bytes	Any data type	Depends on usage.
Short	System.Int16	2 bytes	-32,768 to 32,767	0
Single (single-precision floating-point)	System.Single	4 bytes	-3.402823E38 to -1.401298E-45 for negative values; 1.401298E-45 to 3.402823E38 for positive values	0.0
String	System.String	10 bytes + (2 * string length)	0 to approximately 2 billion Unicode characters	“ ”

Constants

You can use the `Const` statement to declare a constant. Like a variable, a constant holds a value; however, a constant's value is set at design time and may not change. You can include the `Private` or `Public` keyword within the `Const` statement to alter the scope of the constant declaration. Here are three examples:

```
Const Pi As Double = 3.14159
```

```
Private Const CmPerInch As Double = 2.54
```

```
Public Const BookTitle As String = "ASP for Developers"
```

In addition to user-defined constants, VB .NET and the .NET Framework define a number of intrinsic constants. For example, you can use the intrinsic constant `CrLf` anytime you wish to add a carriage return and line feed to a string, as shown in the following snippet.

```
MsgString = "An error has occurred in the program." & _  
CrLf & "Click on OK to continue or CANCEL to abort."
```

NOTE

Most of the old intrinsic constants have changed names in VB .NET. For example, where you would use `vbCrLf` in VB 6.0 and VBScript, you now use `CrLf` in VB .NET.

Implicit and Explicit Variable Declarations

VB has always supported *implicit* variable declarations, which means that you are not required to declare your variables or parameters before using them. However, most professional developers agree that you should not take advantage of this VB feature unless you like bugs in your code. The issue is best demonstrated with an example:

```
Function Multiply(number1, number2)  
    Return number1 * numbr2  
End Function
```

The `Multiply` function in the preceding snippet will always return 0 because we misspelled one of the parameters (`numbr2` instead of the correct `number2`). This happens because VB .NET implicitly declares `numbr2` and initializes it to 0 because it is used in a numeric context. You can avoid this type of hard-to-find bug by using `Option Explicit` or `Option Strict`. In this example, if you had used either of these options, VB .NET would generate a compile-time error when the page was compiled. See the next section for more on `Option Explicit` and `Option Strict`.

Option Explicit versus Option Strict

VB has always had the `Option Explicit` declaration, which forces you to declare all your variables, but VB .NET also introduces `Option Strict`, which goes one step farther. In addition to forcing you to declare all of your variables, `Option Strict` restricts the types of implicit conversions that the language allows. When you use `Option Strict`, VB won't allow conversions where data loss would occur.

To specify `Option Explicit`, you can use the following page directive at the top of the ASP page:

```
<%@ Page Explicit="True" %>
```

To specify `Option Strict`, you can use the following page directive at the top of the ASP page:

```
<%@ Page Strict="True" %>
```

Arrays

You create arrays in VB .NET using the `Dim`, `Public`, or `Private` statements. You use parentheses to specify that you wish to declare an array rather than a scalar variable. For example, the following statement creates an array of strings:

```
Dim Names() As String
```

Before using an array, you must specify the highest element number in the array with the `ReDim` statement:

```
Dim Names() As String  
ReDim Names(1)  
Names(0) = "Mike"  
Names(1) = "Paul"
```

All arrays have a lower bound of zero. The number you place between the parentheses of the `ReDim` statement designates the highest element number the array will hold. Thus, a value of 1 as shown in this example tells VB that the array will hold two string elements, numbered 0 and 1.

Multidimensional Arrays

The `Names` array in the previous example is a one-dimensional array. You can create arrays of multiple dimensions by using commas when you declare the array.

For example, the following statements create a two-dimensional array of strings named `Customer` and a three-dimensional array of double-precision numbers named `Cube`:

```
Private Customer( , ) As String  
Private Cube ( , , ) As Double
```

You would use the following declaration to specify that the `Cube` array would hold $3 \times 3 \times 3 = 27$ elements:

```
ReDim Cube(2,2,2)
```

The following code sets the value of several elements in the `Cube` array:

```
Cube(0,0,0) = 23.4  
Cube(0,0,1) = 14.6  
Cube(2,1,0) = -13.7  
Cube(2,2,2) = 4899.231
```

In addition to using `ReDim`, when declaring an array you can specify the initial size of the array; for example:

```
Dim Names(2) As String
```

Declaring the initial dimensions of an array this way, however, does not restrict you from later resizing the array using `ReDim`.

You can also set the values of an array when you declare it. For example:

```
Dim Names() As String = {"Mike", "Paul"}
```

ReDim and ReDim Preserve

Using `ReDim` erases any existing values of the array. However, you can use the `Preserve` keyword to preserve the values. For example, the following code redimensions the `Colors` array without using the `Preserve` keyword:

```
Dim i As Integer  
Dim Colors(2) As String  
Colors(0) = "Red"  
Colors(1) = "Green"  
Colors(2) = "Blue"  
ReDim Colors(4)  
Colors(3) = "White"  
Colors(4) = "Black"  
  
For i = 0 To UBound(Colors)  
    Response.Write("<br>" & i & "=" & Colors(i))  
Next
```

This produces the following output:

```
0=  
1=  
2=  
3=White  
4=Black
```

NOTE

The For Next loop will be introduced later in this chapter.

Changing the ReDim statement to the following:

```
ReDim Preserve Colors(4)
```

changes the output to:

```
0=Red  
1=Green  
2=Blue  
3=White  
4=Black
```

For multi-dimensional arrays, only the last dimension's values are preserved when using the Preserve keyword.

Checking an Array's Upper Bound

You can use the UBound function to determine the upper boundary of an array. UBound returns the largest available subscript for the array (*not* the number of elements). For example, in the following code, intI = 9 and intJ = 4.

```
Dim intI, intJ As Integer  
Private Square (9 ,4) As Double  
  
intI = UBound(Square, 1)  
intJ = UBound(Square, 2)
```

Passing Parameters

You use parameters to pass information to or from a procedure without having to use global variables. For example, the Divide function has two parameters:

```
Function Divide(Numerator As Double, Denominator As Double) As Double  
    Return Numerator / Denominator  
End Function
```

The Numerator and Denominator variables in this example have been declared with the Double data type.

To declare the data type of the return value of a function, you add As *datatype* to the Function statement after the closing parentheses. In the preceding example, the return value of the Divide function was set to Double. The following function has a return value data type of String:

```
Function SayHello() As String  
    Return "Hello World"  
End Function
```

When you call a procedure, you pass an argument for each parameter. You can specify the list of arguments for a VB procedure in one of two ways: by position or by name. For example:

```
Response.Write(Divide(10,2))
Response.Write("<BR>")
Response.Write(Divide(Denominator:=10, Numerator:=20))
```

The previous example would produce the following output:

5
2

TIP

Although it takes a little more work, your code will be better documented when you pass arguments by name.

By Value versus By Reference

Unlike prior versions of VB, VB .NET, by default, passes arguments to subroutines and functions *by value*. This means that VB sends a copy of each argument's value to the procedure. It also means that parameters, by default, can only be used for input.

You can override this default behavior by using the `ByRef` keyword. When you use `ByRef` when declaring the parameter, VB .NET sends a *pointer*, or reference, to the parameter to the procedure rather than a copy of the parameter's value. Thus, you can use `ByRef` parameters to pass information back to the code that called the procedure.

Optional Parameters

VB .NET supports optional parameters. To create an optional parameter, you insert the `Optional` keyword before the parameter name and you supply the parameter's default value after the data type, like this:

```
Optional parameter_name As data_type = default_value
```

The following function takes a string and makes it into an HTML heading of a level specified by the `Level` parameter. (If `Level` is not specified, it is assumed to be 1.)

```
Function CreateHead(Msg As String, Optional Level As Integer = 1) As String
    Dim ReturnMsg As String
    Dim HLevel As String

    If Level >= 1 And Level <=6 Then
        HLevel = Level.ToString()
        ReturnMsg = "<H" & HLevel & ">" & Msg & "</H" & HLevel & ">"
    Else
        ReturnMsg = "<P>" & Msg & "</P>"
    End If

    Return ReturnMsg
End Function
```

NOTE

The `If` statement will be discussed later in the chapter in the “The If...Then...Else Statement” section. In addition, the `ToString` method, which you can use to convert a number into a string, will be discussed in Chapter 5, “Working with Numbers, String, Dates, and Arrays in Visual Basic .NET.”

The `HelloWorld.aspx` page calls `CreateHead` twice from the `Page_Load` subroutine:

```
<script language="VB" runat="server">
'
Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
        DisplayMsg.Text = CreateHead("Hello World!", 3)
        DisplayMsg.Text &= CreateHead("Hello Universe!")
    End If
End Sub
'
</script>

<asp:label id="DisplayMsg" runat="server" />
```

The first time, the code calls `CreateHead` with the phrase “Hello World” and `Level` equal to 3. The second time, the code calls `CreateHead` with the phrase “Hello Universe” and no specified `Level`, which is interpreted to mean a heading level of 1. This produces a page like the one shown in Figure 4.1.

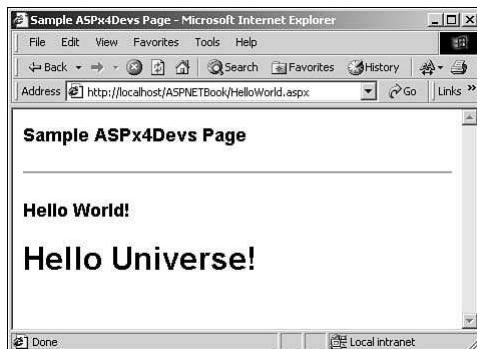


Figure 4.1

This sample page illustrates the use of optional parameters.

Every parameter to the right of an optional parameter must also be optional.

Using Branching and Looping Structures

More than likely, you'll want to be able to conditionally branch in your code based on the value of a variable or an expression. Or perhaps you'll want to repeatedly loop through a section of code. VB .NET supports several branching and looping structures.

Branching in VB .NET

You can branch in your code using the `If...Then...Else` statement or the `Select...Case` statement.

The `If...Then...Else` Statement

You use the `If...Then...Else` statement (or simply the `If` statement) to conditionally execute a piece of code based on the value of some expression. The simplest form of the `If` statement contains an `If` clause without any `Else` clause. Such a statement was used in an earlier example (from `HelloWorld.aspx`):

```
If Not Page.IsPostBack Then  
    DisplayMsg.Text = CreateHead("Hello World!", 3)  
    DisplayMsg.Text &= CreateHead("Hello Universe!")  
End If
```

In this example, the two assignment statements are executed only when `Page.IsPostBack` is `False`. Otherwise, no statements are executed.

NOTE

The `Page` object and the `IsPostBack` property are discussed in Chapter 7, "Understanding ASP.NET Web Forms."

This example, also shown earlier in the chapter, includes an `Else` clause. Any statements in the `Else` block are executed when the `If` expression is `False`:

```
If Level >= 1 And Level <=6 Then  
    HLevel = Level.ToString()  
    ReturnMsg = "<H" & HLevel & ">" & Msg & "</H" & HLevel & ">"  
Else  
    ReturnMsg = "<P>" & Msg & "</P>"  
End If
```

An `If` statement may also contain one or more `Else If` clauses that allow you to test additional scenarios, as shown in this example:

```
If DateTime.Today <= #12/31/2000# Then  
    RegPrice = 895  
ElseIf DateTime.Today <= #01/31/2001# Then  
    RegPrice = 995  
ElseIf DateTime.Today <= #02/28/2001# Then  
    RegPrice = 1095  
Else  
    RegPrice = 1195  
End If
```

The Select...Case Statement

You can also use the `Select...Case` statement for branching in VB .NET. The `Select...Case` statement is useful when you wish to check the value of an expression against a list of possible values and execute a different set of code for each value.

This example checks the value of the integer variable `PayMethod` against a list of possible values. The `Select...Case` statement in the example sets the value of two string variables to various values depending on the value of `PayMethod`.

```
Select Case PayMethod
Case 1
    PayMethText = "Visa"
    SubmitText = "Complete Order and Bill My Credit Card"
Case 2
    PayMethText = "Mastercard"
    SubmitText = "Complete Order and Bill My Credit Card"
Case 3
    PayMethText = "American Express"
    SubmitText = "Complete Order and Bill My Credit Card"
Case 4
    PayMethText = "Company PO"
    SubmitText = "Complete Order"
Case 5
    PayMethText = "Check"
    SubmitText = "Complete Order"
Case Else
    PayMethText = "Error"
    SubmitText = "Illegal Payment Method: please correct."
End Select
```

Notice the `Case Else` clause in the preceding listing, which is executed if none of the other cases is true.

Looping in VB .NET

You can loop using the `Do...Loop` statement, the `While...End While` statement, the `For...Next` statement, or the `For...Each` statement.

The Do...Loop Statement

You can use the `Do...Loop` statement (or simply `Do loop`) to execute a set of statements repeatedly, either while some condition is true or until some condition becomes true.

For example, the following code from `titles.aspx` fills a `dropdownlist` control with records from the `titles` table of the `pubs` sample SQL Server database. We have used a `Do loop` to move through each of the records returned by the query, and added them to the `dropdownlist`'s `ListItem` collection. (See Chapter 10, “Designing Advanced Interfaces with Web Form List Controls,” for more on Web Form list controls, and see Chapter 14, “Accessing Data with .NET Data Providers,” for more on using ADO.NET with the SQL Managed Provider.)

```
Sub FillList()
    Dim ConnectString As String = _
        "server=localhost;uid=sa;pwd=;database=pubs"
    Dim Sql As String
    Dim PubsCnx As SqlConnection
    Dim TitlesQry As SqlCommand
    Dim TitlesRdr As SqlDataReader
    Dim TitleItem As ListItem

    PubsCnx = New SqlConnection(ConnectionString)
    PubsCnx.Open()
    Sql = "SELECT title, title_id FROM titles ORDER BY title"
    TitlesQry = New SqlCommand(Sql, PubsCnx)
    TitlesRdr = TitlesQry.ExecuteReader()

    Do Until Not TitlesRdr.Read()
        TitleItem = New ListItem(TitlesRdr("title"), TitlesRdr("title_id"))
        TitleList.Items.Add(TitleItem)
    Loop
End Sub
```

The Do...Loop statement from the `FillList` subroutine uses the `SqlDataReader`'s `Read` method to retrieve the next record returned by the query. `Read` advances the current record pointer and returns `True` if it was able to successfully retrieve a record or `False` if there are no more records to retrieve.

This Do...Loop statement could have been rewritten by reversing the logic of the condition:

```
Do While TitlesRdr.Read()
    TitleItem = New ListItem(TitlesRdr("title"), TitlesRdr("title_id"))
    TitleList.Items.Add(TitleItem)
Loop
```

The VB .NET Do...Loop statement also supports testing the condition at the bottom of the loop. For example, the following loop squares itself until the result exceeds 100:

```
Dim x As Integer = 2
Do
    x = x*x
Loop Until x > 100
```

You can use the `Exit Do` statement to terminate a `Do` loop early. For example, the following code (from `DotSearch.aspx`) searches for the position of the first period in a string, exiting the `Do` loop when a period is found:

```
Dim Phrase As String = "My mother is very, very smart. My father is too."
Dim Length As Integer = Len(Phrase)
Dim i As Integer = 1
```

```
Do While i<=Length
    If Phrase.Chars(i) = "." Then
        Exit Do
    End if
    i += 1
Loop
' Add one to i because the first character
' in a string is character 0 in VB.
Label1.Text = "The first period in '" & Phrase &
"' was found at character " & i + 1 & "."
```

NOTE

The preceding example was provided for demonstration purposes only. You should use the .NET Framework `String.IndexOf` method to search for strings within other strings.

The While...End While Statement

The `While...End While` statement (or simply the `While` loop) is very similar to the `Do...Loop` statement. You can use it to execute a set of statements repeatedly, for as long as some condition is true. For example:

```
While i<=Length
    If Mid(Phrase, i, 1) = "." Then
        Exit While
    End if
    i += 1
End While
```

As shown in the preceding example, you can use the `Exit While` statement to terminate a `While` loop early.

NOTE

The `While...Exit While` statement replaces the old `While...Wend` statement of earlier versions of VB.

The For...Next Statement

You can use the `For...Next` statement (or simply the `For` loop) to repeatedly execute a block of statements a specified number of times. While similar in concept to the `Do` and `While` loops, the `For` loop differs in that it automatically increments a counter variable for you.

The `For` loop is especially useful for iterating through the items in an array. For example, the following code iterates through all of the elements in the `Colors` array and displays them on the page:

```
For i = 0 To UBound(Colors)
    Response.Write("<br />" & i & "=" & Colors(i))
Next
```

The For...Each Statement

The For...Each statement is a special kind of For...Next loop that is useful for iterating through members of a *collection*. A collection is an ordered set of items, usually objects, that you can refer to and manipulate as a unit. For example, when working with ADO.NET, you can work with the Errors collection of Error objects.

NOTE

You can use the Collection object in VB .NET to create your own collections.

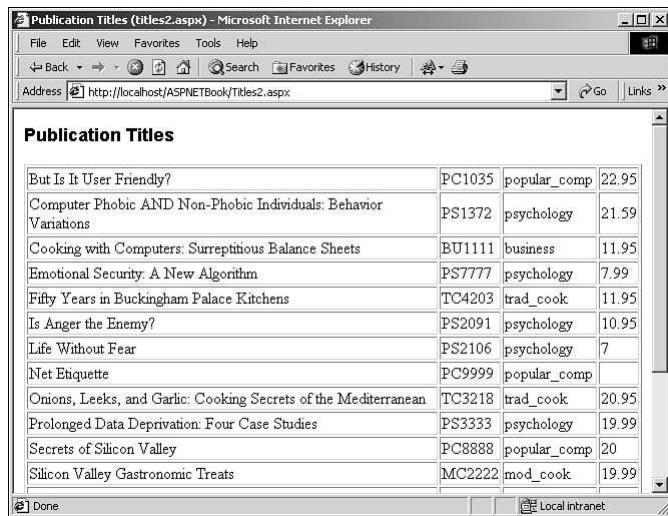
For example, the following function (from titles2.aspx) returns an HTML table containing a row for each record in the titles table of the SQL Server Pubs database. The dataset's Table object contains a collection of rows, and each row contains a collection of columns. The DisplayTitles function employs two nested For...Each loops to iterate through the TitlesSet dataset. (Datasets and ADO.NET are explained in more detail in Chapter 15, “Working with ADO.NET DataSets.”)

```
Function DisplayTitles()
    Dim ConnectString As String = _
        "server=localhost;uid=sa;pwd=;database=pubs"
    Dim Sql As String
    Dim PubsCnx As SqlConnection
    Dim TitlesQry As SqlDataAdapter
    Dim TitlesSet As DataSet
    Dim TitleItem As ListItem
    Dim row As DataRow
    Dim col As DataColumn
    Dim Msg As String

    PubsCnx = New SqlConnection(ConnectionString)
    PubsCnx.Open()
    Sql = "SELECT title, title_id, type, price FROM titles ORDER BY title"
    TitlesQry = New SqlDataAdapter(Sql, PubsCnx)
    TitlesSet = New DataSet
    TitlesQry.Fill(TitlesSet, "titles")

    Msg &= "<TABLE border=""1"">"
    For Each row In TitlesSet.Tables("titles").Rows
        Msg &= "<TR>"
        For Each col In TitlesSet.Tables("titles").Columns
            Msg &= "<TD>" & FixNull(row(col), "&nbsp;") & "</TD>"
        Next
        Msg &= "</TR>"
    Next
    Msg &= "</TABLE>"
    Return Msg
End Function
```

The titles2.aspx page is shown in Figure 4.2.



The screenshot shows a Microsoft Internet Explorer window with the title "Publication Titles (titles2.aspx) - Microsoft Internet Explorer". The address bar shows the URL "http://localhost/ASPNETBook/Titles2.aspx". The main content area displays a table titled "Publication Titles" with 15 rows of data. The columns represent book titles, their ID numbers, categories, and prices.

Publication Titles			
But Is It User Friendly?	PC1035	popular_comp	22.95
Computer Phobic AND Non-Phobic Individuals: Behavior Variations	PS1372	psychology	21.59
Cooking with Computers: Surreptitious Balance Sheets	BU1111	business	11.95
Emotional Security: A New Algorithm	PS7777	psychology	7.99
Fifty Years in Buckingham Palace Kitchens	TC4203	trad_cook	11.95
Is Anger the Enemy?	PS2091	psychology	10.95
Life Without Fear	PS2106	psychology	7
Net Etiquette	PC9999	popular_comp	
Onions, Leeks, and Garlic: Cooking Secrets of the Mediterranean	TC3218	trad_cook	20.95
Prolonged Data Deprivation: Four Case Studies	PS3333	psychology	19.99
Secrets of Silicon Valley	PC8888	popular_comp	20
Silicon Valley Gastronomic Treats	MC2222	mod_cook	19.99

Figure 4.2

Nested For...Each *loop* are used to display titles from the *Pubs* database.

Creating Objects

Prior versions of VB lacked many object-oriented programming (OOP) features that other languages such as C++, Java, and FoxPro have had for years. Fortunately, VB .NET includes strong support for OOP.

OOP Primer

Class, subclass, inheritance, constructor, polymorphism—object-oriented programming uses lots of fancy new terms that undoubtedly confuse the non-OOP programmer. This section provides a 10-minute primer of OOP terminology.

Objects and Classes

Objects are things that you want to represent in your code. Another way to think of an object is as a grouping of properties, methods, and events that are logically tied together. You work with an object by manipulating its properties and methods and reacting to its events.

A *class* is a template or schema for creating an object. At design time you create the class that serves as the template for creating objects at runtime. An object is thus an instance of a class. And that's one of the neat things about using classes: You can have as many instances of a class as you want, and VB automatically keeps each object's data independent of each other object's data.

Another cool thing about classes is that they encapsulate the implementation of the object into a neat package. *Encapsulation* allows you to separate the implementation of the class (the code inside of the class that makes it work) from its interface (the public properties, methods, and events of the class).

Inheritance and Polymorphism

One of the big additions to VB .NET is its support for inheritance. *Inheritance* allows you to create classes that are descendants of another class. When a class inherits from another class, the original class is termed the *base class* (also sometimes referred to as the superclass or parent class) and the class that inherits from the base class is called the *derived class* (also sometimes referred to as the subclass or child class).

VB .NET supports the *overriding* of a base class's methods with alternate implementations. *Polymorphism* is the ability of different classes to support properties and methods with the same name but with different implementations. VB .NET's support for overriding allows your classes to support polymorphism.

Creating a Class

You use the `Class` statement in VB .NET to create a class. Public variables of the class become properties of the class and public subroutines or functions of the class become methods.

Let's illustrate with an example: Say that your Web site supports an ASP.NET conference called "ASP.NET—The Conference." To simplify the registration process, you decide to create a `Register` class. The `Register` class will have two properties that describe the attendee, `CustomerFirstName` and `CustomerLastName`, and two methods, `Execute` and `Cancel`.

```
Class Register
    Public CustomerFirstName As String
    Public CustomerLastName As String

    Public Function Execute() As String
        Return "<BR /><font color=""green"">Register.Execute</font>""
    End Function

    Public Function Cancel() As String
        Return "<BR /><font color=""red"">Register.Cancel</font>""
    End Function
End Class
```

To create an object from a class, you use the `New` operator just like when creating VB and .NET Framework objects. The following code declares `Reg` as a member of the `Register` class and instantiates `Reg`:

```
Dim Reg As Register = New Register
```

The preceding can be simplified as follows:

```
Dim Reg As = New Register
```

The following code sets two properties of the `Reg` object:

```
Reg.CustomerFirstName = "Bill"  
Reg.CustomerLastName = "Gates"
```

This code invokes the `Execute` method of `Reg`, setting the `Text` property of a label control to its return value:

```
Label1.Text = Reg.Execute()
```

While you could also have programmed the registration process with a series of functions and subs, the beauty of using classes is that you encapsulate all of the details associated with registering an attendee, wrapping it all up into a class with a very simple interface. Of course, either way, you have to write the code that does the work. But by coding the registration process using classes, you benefit from improved reusability, manageability, and extensibility.

Using Property Statements

There are actually two ways to create properties of classes in VB .NET. You've already seen one way to create properties: create public variables of the class module. A second way to create properties is to use the `Property` statement. Using a `Property` statement to create properties offers several advantages over using public variables. Using a `Property` statement lets you

- Create read-only and write-only properties
- Run code when a property value is read or written

The `Property` statement must follow a certain format. For read/write properties, it should look like this:

```
Property property_name As data_type  
    Get  
        property_name = some_value  
    End Get  
    Set  
        some_variable = Value  
    End Set  
End Property
```

The `Value` keyword within a `Set` block retrieves the value to which the consumer of the class has set the property.

Read-only properties must include the `ReadOnly` keyword and have no `Set` block:

```
ReadOnly Property property_name As data_type  
    Get  
        property_name = some_value
```

```
    End Get  
End Property
```

Write-only properties must include the `WriteOnly` keyword and have no `Get` block:

```
WriteOnly Property property_name As data_type  
    Set  
        some_variable = Value  
    End Set  
End Property
```

The following class (from `Register2.aspx`) uses `Property` statements to create four properties. The `CustomerFirstName` and `CustomerLastName` properties are read/write. `CustomerName` is read-only and `CreditCardNumber` is write-only:

```
Class Register  
    Private CustFname As String  
    Private CustLName As String  
    Private FullName As String  
    Private CreditCard As String  
  
    Public Property CustomerFirstName As String  
        Get  
            CustomerFirstName = CustFName  
        End Get  
        Set  
            CustFName = Value  
            FullName = CustFName & " " & CustLName  
        End Set  
    End Property  
  
    Public Property CustomerLastName As String  
        Get  
            CustomerLastName = CustLName  
        End Get  
        Set  
            CustLName = Value  
            FullName = CustFName & " " & CustLName  
        End Set  
    End Property  
  
    Public ReadOnly Property CustomerName As String  
        Get  
            CustomerName = FullName  
        End Get  
    End Property
```

```

Public WriteOnly Property CreditCardNumber As String
    Set
        CreditCard = Value
    End Set
End Property

Public Function Execute() As String
    Return "<BR /><font color=""green"">Register.Execute</font>""
End Function

Public Function Cancel() As String
    Return "<BR /><font color=""red"">Register.Cancel</font>""
End Function
End Class

```

NOTE

The methods in these demonstration examples do nothing but return some text.

You'll get a compile error if you try to write to a read-only property or read a write-only property. The Register2a.aspx page illustrates what happens when trying to read a write-only property (see Figure 4.3).

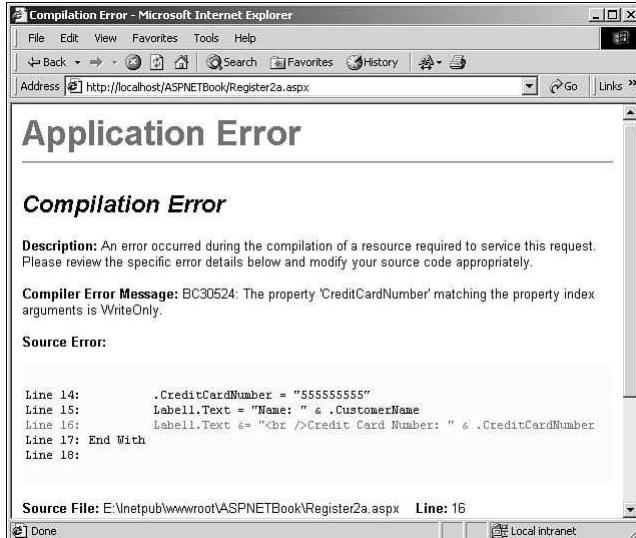


Figure 4.3

Attempting to read a write-only property causes a compilation error.

NOTE

In versions of VB before VB .NET, you used **Property** procedures instead of VB .NET's **Property** statement to create properties.

Inheritance

VB .NET provides the **Inherits** statement, which allows you to create classes that derive from other classes.

WARNING

This chapter makes inheritance look simple, but in reality it's not. Whole books have been written on the issues of inheritance and application design. Inheritance should be used with caution.

Let's say that you decide to branch out and run a conference on XML in addition to ASP.NET, but the registration process for XML is different enough that you need to make some changes to the process.

To handle the new show, you create two subclasses of **Register**: **ASPRregister** and **XMLRegister**. For **XMLRegister**, you add an additional property—**UserGroupAffiliation**—because the XML conference will give special discounts to qualified user-group members. Without inheritance, you would have to either change the base class or copy and paste code from class to class. Inheritance, however, allows you to start with the existing **Register** class and extend it with additional properties and methods. But there's no need to re-create the properties and methods that **XMLRegister** has in common with the base class.

In the following code from **Register3.aspx**, the **ASPRregister** and **XMLRegister** classes are derived from the **Register** base class:

```
Class Register
    Public CustomerFirstName As String
    Public CustomerLastName As String
    Public CreditCardNumber As String
    Public CreditCardExpires As String

    Public Function Execute() As String
        Return "<BR /><font color=""green"">Register.Execute</font>""
    End Function

    Public Function Cancel() As String
        Return "<BR /><font color=""red"">Register.Cancel</font>""
    End Function
End Class

Class ASPRegister
    Inherits Register
End Class
```

```

Class XMLRegister
    Inherits Register
    Public UserGroupAffiliation As String
End Class

```

The following code from Register3.aspx instantiates both ASPRegister and XMLRegister objects:

```

Sub Page_Load(Src as Object, E as EventArgs)
    Dim ASPReg As ASPRegister
    Dim XMLReg As XMLRegister

    ASPReg = New ASPRegister
    With ASPReg
        .CustomerFirstName = "Bill"
        .CustomerLastName = "Gates"
        .CreditCardNumber = "555555555555"
        .CreditCardExpires = "01/02"
        Label1.Text = .Execute()
    End With

    XMLReg = New XMLRegister
    With XMLReg
        .CustomerFirstName = "Steve"
        .CustomerLastName = "Balmer"
        .CreditCardNumber = "555555555556"
        .CreditCardExpires = "05/03"
        .UserGroupAffiliation = "Microsoft Windows 2000 UG"
        Label2.Text = .Execute()
    End With

    ASPReg = Nothing
    XMLReg = Nothing
End Sub

```

Accessibility of Inherited Properties and Methods

When creating classes, you use the `Public` keyword to create publicly exposed properties and methods of the object. If you don't want a class method or property to be public, however, you have a few other choices: `Private`, `Protected`, `Friend`, or `Protected Friend`.

Use the `Private` keyword to make a property or method private to the class. Private properties and methods can be accessed anywhere inside of the class but they can't be accessed by derived classes or other classes on the page or assembly (assemblies are discussed in Chapter 6, “Using Namespaces and Assemblies with Visual Basic”).

You use the `Protected` keyword to create a property or method that is accessible from any code inside the class and from code inside of any derived classes.

Use the `Friend` keyword to create a property or method that is accessible from any code—including derived classes—in the current page or assembly. `Friend` properties and methods are not accessible from derived classes that live in other assemblies.

Finally, you can use the `Protected Friend` keyword to create a property or method that's available from any code in the current page or assembly *and* from any derived classes, no matter where they reside.

Overriding Methods

If you wish to override a method of the base class, you use the `Overrides` keyword.

Let's say you add a third conference, one on Linux, but the Linux conference is different from the other two because registration is free. You can create a third subclass, `LinuxRegister`, but this time you wish to change the `Execute` method. You can override the `Execute` method from the base class (`Register`), and replace the implementation of `Execute` with a new implementation that is custom-tailored for the Linux show.

Here's the code from `Register4.aspx` that overrides the `Register.Execute` method:

```
Class LinuxRegister
    Inherits Register
    Overrides Public Function Execute() As String
        Return "<BR /><font color=""blue"">Register.Execute</font>" 
    End Function
End Class
```

This code won't work, however, unless the `Execute` method is marked as a method that can be overridden back in the base class. To do that, add the `Overridable` keyword to the declaration of the method, in the base class. Here's the `Register` class from `Register4.aspx`:

```
Class Register
    Public CustomerFirstName As String
    Public CustomerLastName As String
    Public CreditCardNumber As String
    Public CreditCardExpires As String

    Overridable Public Function Execute() As String
        Return "<BR /><font color=""green"">Register.Execute</font>" 
    End Function

    Overridable Public Function Cancel() As String
        Return "<BR /><font color=""red"">Register.Cancel</font>" 
    End Function
End Class
```

Notice that we have marked the `Execute` and `Cancel` functions as `Overridable`. Methods in VB .NET, by default, are *not* overridable (you can also explicitly include the `NotOverridable` keyword when you define the method, but it's not necessary).

If you wish to access a base method from a derived method, you can use the MyBase pseudo-object. For example, if you wished to call the Register class's Execute function from the LinuxRegister Execute method, you would use code like this:

```
Class LinuxRegister
    Inherits Register
    Overrides Public Function Execute() As String
        MyBase.Execute()
    End Function
End Class
```

Constructing an Object

VB .NET lets you create a subroutine that is executed when an object is instantiated.

A *class constructor* is called when an object is instantiated. To create a constructor, you add a subroutine named New to the class definition (you can place it anywhere in the class definition). The class constructor subroutine will be called when an object is instantiated before any other code in the class is executed. Class constructors may have parameters—these types of constructors are called *parameterized constructors*. Parameterized constructors allow you to pass parameters to a class when the object is instantiated.

The Register class from Register5.aspx contains a parameterized class constructor that you can use to optionally set the customer name when instantiating the class:

```
Sub New(Optional LastName As String = "", Optional FirstName As String = "")
    CustLName = LastName
    CustFName = FirstName
    FullName = CustFName & " " & CustLName
End Sub
```

Here's code from the two derived classes that also contain class constructors:

```
Class ASPRegister
    Inherits Register

    Sub New(Optional LastName As String = "", Optional FirstName As String = "")
        MyBase.New(LastName, FirstName)
    End Sub
End Class

Class XMLRegister
    Inherits Register

    Public UserGroupAffiliation As String

    Sub New(Optional LastName As String = "", Optional FirstName As String = "")
        MyBase.New(LastName, FirstName)
        UserGroupAffiliation = "none"
    End Sub
End Class
```

Notice the use of the `MyBase.New`. Derived classes need to explicitly call the `MyBase.New` constructor or the base class constructor code will not run.

Here's the code from `Register5.aspx` that instantiates the `ASPRegister` and `XMLRegister` objects, passing along the optional customer name information to the objects' constructors:

```
ASPReg = New ASPRegister("Gates", "Bill")
XMLReg = New XMLRegister("Balmer", "Steve")
```

Summary

This chapter introduced the Visual Basic .NET language and discussed its core features, including operators, procedures, variables, parameters, data types, constants, and branching and looping statements. The chapter also introduced object-oriented programming (OOP) concepts and VB .NET OOP features, including classes, properties, methods, inheritance, and overriding.

VB .NET is an exciting new release of the venerable Visual Basic language, and a language that is now on equal footing when compared to Visual C++ and C#. Microsoft has added a number of new features to the language that will undoubtedly be appreciated by the vast majority of existing VB and ASP programmers—who will quickly forget about VB 6.0 and VBScript.

CHAPTER 5

Working with Numbers, Strings, Dates, and Arrays in Visual Basic .NET

In the evolution from Visual Basic (VB) to Visual Basic .NET (VB .NET), Microsoft decided to take a lot of the functionality that was historically in the VB language (and various Visual C++ libraries) and move it into the .NET Framework. In this chapter, you'll learn how to use a number of the .NET Framework System classes to manipulate numbers, strings, and dates. You'll also look at some of the array-handling features that the System classes provide.

NOTE

VB .NET is currently in transition, because some of the old ways of doing things are still there in the VB .NET language alongside the new System class functionality.

This chapter doesn't discuss the old ways—you'd be advised to avoid the older functionality as much as possible. This chapter focuses on the new .NET Framework functionality exclusively. If you are upgrading legacy applications that contain code that uses the old VB functionality, you may want to look at Appendix A, "Moving to VB .NET from VB6 or VBScript," which discusses migration issues.

Using the System Classes

As you learned in Chapter 3, “What’s New with ASP.NET?,” the .NET Framework provides a lot of functionality that is available from any .NET programming language, including Visual Basic .NET. The .NET Framework makes this functionality available in the form of a number of system classes.

Like all .NET Framework classes, the system classes live within *assemblies*. Although it’s an oversimplification, you can think of assemblies as the .NET equivalent of DLL and EXE files. In order to use the functionality inside of an assembly, you must link the assembly to your page. Fortunately, ASP.NET (by virtue of entries made in the machine configuration file, `Machine.Config`) automatically includes a number of core classes—including those discussed in this chapter—into the Web application without you having to do a thing.

In addition to it being physically part of an assembly, a class is also logically part of a *namespace*. For example, the `Math` class is part of the `System` namespace, even though it lives inside of the `mscorlib.dll` assembly.

NOTE

Chapter 6, “Using Namespaces and Assemblies with Visual Basic .NET,” discusses assemblies in detail.

Instance and Static Class Members

Classes contain *members*. Class members include fields, properties, and methods. Properties and methods were discussed in Chapter 4, “Understanding Visual Basic .NET Syntax and Structure.” *Fields* are very similar to properties—for all practical purposes you can consider the two to be the same.

Members can be of two basic types: instance or static. An *instance member* is a property, field, or method that is private to a particular instance of an object. For example, the `Length` property of the `System.String` class is an instance member. You might use the following code to calculate the length of the string variable, `Stuff`:

```
Dim Stuff As String = "somedata"
Answer = Stuff.Length
```

A *static* (or *shared*) *member* is a property, field, or method that is shared across all instances of a class. This allows you to use the member without having to instantiate the object. For example, to calculate the square root of a number you can use the `System.Math` class’s `Sqrt` method like this:

```
Answer = Math.Sqrt(81)
```

Because `Sqrt` is a static method, you don’t actually need to instantiate a `Math` object to use it. Instead, you use the name of the class as the object. Many of the `System` classes contain static members, which are useful for obtaining utility-type functionality.

Working with Numbers

The `System.Math` class provides a number of mathematical functions you can use to make trigonometric, logarithmic, and other mathematical calculations.

E and PI

The `System.Math` class contains two static fields, `E` and `PI`, which return the value of `e` and `pi`, respectively.

For example, the `Calc` subroutine in Listing 5.1 calculates the circumference of a circle using the `Math.PI` field. In this example, the value of the radius is obtained from the value of a text box control, `radiusbox`.

Listing 5.1 CIRCLE1.ASPX—This Subroutine Makes Use of Math.PI to Calculate the Circumference of a Circle

```
Function Calc()
    Dim Circum As Double
    Dim Radius As Double

    Radius = Convert.ToDouble(radiusbox.Text)
    Circum = Math.PI * 2 * Radius
    Return "<i>Circumference: " & Circum & "</i>"
End Function
```

Trigonometry

The `System.Math` class contains a number of static methods for performing trigonometric calculations. You can calculate the sine, cosine, tangent, arcsine, arccosine, arctangent, hyperbolic sine, hyperbolic cosine, and hyperbolic tangent using methods of `System.Math`.

For example, you can use the `Sin` method to calculate the sine of an angle of a right triangle. You pass `Sin` the angle in radians and it returns the ratio of the length of the side opposite to the angle divided by the length of the hypotenuse. The following code calculates the sine of a 30-degree angle:

```
Result = Math.Sin(30*Math.PI/180)
```

TIP

To convert an angle in degrees to radians, you multiply the angle by pi and divide by 180.

Rounding and Truncation

The `System.Math` class contains several static methods that are useful for rounding or truncating numbers. To round a real number, you can use `Round`. `Round` works with either a `Double` or `Decimal` value, returning the rounded number using the same data

type passed to Round. Round returns the whole number nearest to the real number. If the real number is exactly halfway between two whole numbers, Round returns the even whole number.

The Ceiling and Floor methods of System.Math provide an alternative to Round. Floor returns the largest whole number smaller than or equal to the passed value. For positive numbers, Floor truncates the value. Ceiling returns the smallest whole number larger than or equal to the passed value. For positive numbers, Ceiling rounds the value up. For negative numbers, however, Floor and Ceiling have the opposite effect of what you might expect. For example, the Floor and Ceiling of 48.7 are 48 and 49, respectively, whereas the Floor and Ceiling of -48.7 are -49 and -48, respectively. Ceiling and Floor both take a Double parameter and return a Double.

Listing 5.2 illustrates the differences between Round, Ceiling, and Floor:

Listing 5.2 ROUND.ASPX—This Code Demonstrates the Use of the Match Class's Round, Ceiling, and Floor Methods

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim DblNumber() As Double = {1.0004, 35.4, 35.5, 35.6, -48.4, -48.5,
    ➔-48.6, 99.99}
    Dim i As Integer

    output.Text &= "<table border=""1"">" &
        "<tr><th>Real Number</th><th>Math.Round</th>" &
        "<th>Math.Ceiling</th><th>Math.Floor</th></tr>"

    For i = 0 To DblNumber.GetUpperBound(0)
        output.Text &= "<tr align=""right"">"
        output.Text &= "<td>" & DblNumber(i) & "</td>"
        output.Text &= "<td>" & Math.Round(DblNumber(i)) & "</td>"
        output.Text &= "<td>" & Math.Ceiling(DblNumber(i)) & "</td>"
        output.Text &= "<td>" & Math.Floor(DblNumber(i)) & "</td>"
        output.Text &= "</tr>"

    Next

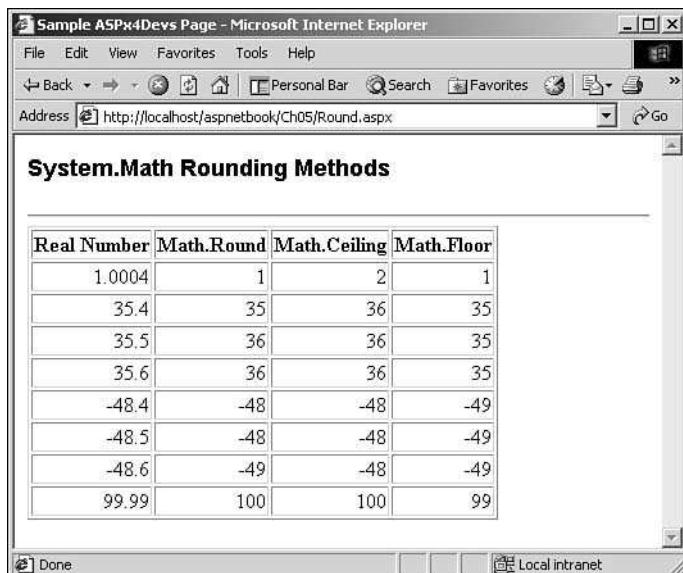
    output.Text &= "</table>"
End Sub
```

The output produced by Round.aspx is shown in Figure 5.1.

Powers, Roots, and Logarithms

System.Math provides a few methods for the calculation of powers, roots, and logarithms. You can use the Pow method to raise a number to a specified power. For example, the following code raises 2 to the power of 8 (that is, multiplies 2 by itself 8 times):

```
Result = Math.Pow(2,8) '= 256
```

**Figure 5.1**

The System.Math methods for rounding and truncation.

The `Sqrt` method calculates the square root of a number. For example, the following code calculates the square root of 237:

```
Result = Math.Sqrt(237) '= 15.3948043183407
```

The `Exp` method raises e (the natural exponent) to a power. For example, the following code raises e to the 5th power:

```
Result = Math.Exp(5) '= 148.413159102577
```

You can use the `Log` method to reverse the action of `Exp`, returning the natural logarithm of a number. For example:

```
Result = Math.Log(148.413159102577) '= 5
```

The `Log10` method returns the base 10 logarithm of a number, that is, it returns the number to which 10 would have to be raised to get a particular value. For example, the following code calculates the power to which 10 would have to be raised to return one million:

```
Result = Math.Log10(1000000) '= 6
```

Miscellaneous Mathematical Functionality

`System.Math` contains a few other methods that you may find useful. The `Abs` method returns the absolute value of a number, that is, the number without any sign. A close relative of `Abs` is the `Sign` method, which returns 1 if the number is greater than zero, 0 if the number is equal to zero, and -1 if the number is less than zero. For example:

```
Result = Math.Abs(-73.8)  '= 73.8
```

```
Result = Math.Sign(-73.8) '= -1
```

You can use the `Min` and `Max` methods to return the minimum and maximum of two numbers. For example:

```
Result = Math.Min(3, 5) '= 3
```

```
Result = Math.Max(3, 5) '= 5
```

Manipulating Strings

The `System.String` class provides a great deal of functionality for the manipulation of strings. In many cases, the properties and methods of the `String` class duplicate functionality that VB .NET provides. For example, you can obtain the length of a string by using either the VB `Len` function or the `String.Length` property. In general, your code will be better suited for the future if you use the new `System` classes rather than the older VB functionality.

The `String` class contains both static and instance members. To use a static member such as the `Compare` method, you use `String` as the object. Here's an example of using the static `Compare` method to compare two strings:

```
Result = String.Compare("abc", "abcd")
```

You can use the `String` class's instance members on any string variable or even on literal strings. For example:

```
Result = myString.Length
```

```
Result = "My favorite color is blue".Length
```

In some cases, the `String` class provides both static and instance methods that offer similar functionality. For example, you can compare two strings using either the static method `Compare` or the instance method `CompareTo`. The two lines of code below are equivalent:

```
Result1 = String.Compare("abc", "abcd")
```

```
Result2 = "abc".CompareTo("abcd")
```

Determining a String's Length

The `String` class's `Length` property returns the length of a string. For example:

```
Dim Sentence As String = "My favorite color is blue."
Result = Sentence.Length '= 26
```

Searching for and Extracting Substrings

You can use the `IndexOf`, `IndexOfAny`, `LastIndexOf`, `LastIndexOfAny`, `StartsWith`, and `EndsWith` methods to search for a substring within a string. All of these methods work on a particular instance of a string.

You can use the `IndexOf` method to search for a string, a character, or an array of characters within another string. You can pass `IndexOf` an argument of type `String` or `Char` and it returns the starting position of the string or character within the original string, or -1 if the string is not found. You can pass `IndexOfAny` an array of type `Char` and it returns the first position of any characters within the array, or -1 if none of the characters is found. `IndexOf` and `IndexOfAny` searches are case-sensitive.

Listing 5.3, from `IndexOf.aspx`, demonstrates the use of `IndexOf` and `IndexOfAny` to search for a string, a character, and an array of characters.

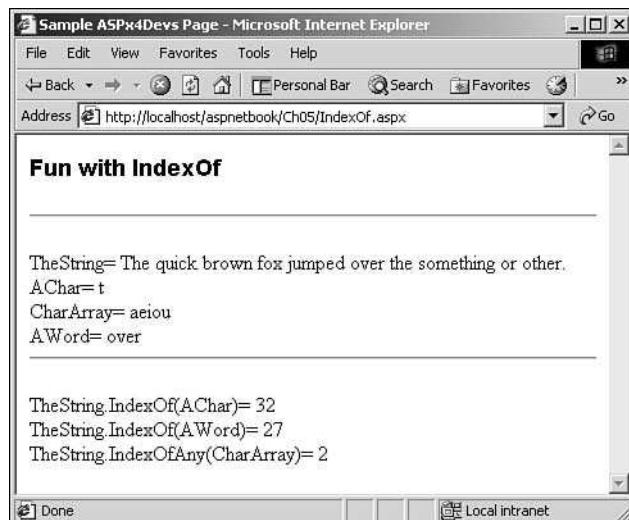
Listing 5.3 INDEXOF.ASPX—This Code Demonstrates the Use of the IndexOf and IndexOfAny Methods

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim TheString As String = _
        "The quick brown fox jumped over the something or other."
    Dim AChar As Char = "t"c
    Dim AWord As String = "over"
    Dim CharArray() As Char = {"a"c, "e"c, "i"c, "o"c, "u"c}
    Dim i As Integer

    output.Text &= "<br />TheString= " & TheString
    output.Text &= "<br />AChar= " & AChar
    output.Text &= "<br />CharArray= "
    For i = 0 To CharArray.GetUpperBound(0)
        output.Text &= CharArray(i)
    Next
    output.Text &= "<br />AWord= " & AWord

    output.Text &= "<hr />"
    output.Text &= "<br />TheString.IndexOf(AChar)= " & _
        TheString.IndexOf(AChar)
    output.Text &= "<br />TheString.IndexOf(AWord)= " & _
        TheString.IndexOf(AWord)
    output.Text &= "<br />TheString.IndexOfAny(CharArray)= " & _
        TheString.IndexOfAny(CharArray)
End Sub
```

You can see the output from `IndexOf.aspx` in Figure 5.2.

**Figure 5.2**

You can use the `IndexOf` and `IndexOfAny` methods of the `String` class to search for substrings of type `String`, `Char`, and array of `Char`.

TIP

You use “`x`”`c` to represent a `Char` literal.

`IndexOf` has two optional `Integer` parameters that you can use to specify the starting and ending position of the search. For example, you could use the following code to search `TheString` for the word “the” from position 9 until the end of the string:

```
Result = TheString.IndexOf("the", 9)
```

To search `TheString` for the word “and” from position 4 through position 24, you would use the following code:

```
Result = TheString.IndexOf("and", 4, 24)
```

`IndexOfAny` also lets you limit the search using optional `Integer` parameters.

NOTE

The `LastIndexOf` and `LastIndexOfAny` methods work similarly to the `IndexOf` and `IndexOfAny` methods, except that they return the position of the last occurrence of the search string.

The `StartsWith` method takes a parameter of type `String` and returns `True` if the string begins with that substring, or `False` if it does not. Similarly, the `EndsWith` method returns whether or not a string ends with a particular substring. `StartsWith` and `EndsWith`, like `IndexOf`, `IndexOfAny`, `LastIndexOf`, and `LastIndexOfAny` perform case-sensitive matches.

The following code illustrates how you might use `StartsWith` and `EndsWith` to determine if `DBField` started with or ended with “`Id`”:

```
Result = DBField.StartsWith("Id") Or DBField.EndsWith("Id")
```

To retrieve a single character from a string, you can use the `Chars` property. For example, the following code retrieves the 4th character (“`f`”) from the `Sentence` string (the first character in a string is character 0):

```
Dim Sentence As String = "My favorite color is blue."
Result = Sentence.Chars(3) '= "f"
```

You can use the `Substring` method to retrieve a substring from a string. You pass `Substring` the starting position and an optional length. For example, the following code uses `Substring` to retrieve the substring “`quick`” from `TheString`:

```
Dim TheString As String =
    "The quick brown fox jumped over the something or other."
Result = TheString.Substring(4, 5) '= "quick"
```

If you leave off the second parameter, `Substring` grabs the substring from the starting position to the end of the string. For example, the following code uses `Substring` to retrieve the substring “`other.`” from `TheString`:

```
Result = TheString.Substring(49) '= "other."
```

You can use the `Replace` method to replace all occurrences of a particular character with another. `Replace` takes two parameters, both of type `Char`: the existing character to replace, and the character with which to replace it. The following code replaces every “`e`” in `TheString` with an “`x`”:

```
TheString = TheString.Replace("e", "x")
```

After running this code, `TheString` becomes:

```
Thx quick brown fox jumpxd ovxr thx somxthing or othxr.
```

Splitting and Joining Strings

You can use the `String` class's `Split` method to split a string into an array of substrings. You supply `Split` with a separator character of type `Char` and it creates an array of type `String`. For example, the following code creates an array of the words in `TheString`:

```
Dim TheString As String = "ASP.NET for Developers"  
Dim TheWords() As String  
TheWords = TheString.Split("c")
```

Here are the elements of `TheWords()` after running this code:

```
{"ASP.NET", "for", "Developers"}
```

The `Join` method, a static method of the `String` class, does the opposite of `Split`. It takes a separator string of type `String` and an array of strings and turns it into a string, separating each array element from other elements using the separator character. For example, you could use the following code to reverse the action of the `Split` method shown in the previous example, placing the joined string into the `JoinedString` variable:

```
Dim JoinedString As String  
JoinedString = String.Join(" ", TheWords)
```

After running this code, `JoinedString` equals:

```
ASP.NET for Developers
```

The `Concat` method (also a static method) overlaps the functionality of `Join`. You can use `Concat` to create a string from an array of type `String`. Unlike `Join`, however, `Concat` joins the array elements together without using a separator character. For example, the following code uses `Concat` to join together the elements of the array `TheWords`:

```
Dim ConcatString As String  
ConcatString = String.Concat(TheWords)
```

After running this code, `ConcatString` equals:

```
ASP.NETforDevelopers
```

Trimming and Padding

You can use the `Trim`, `TrimStart`, or `TrimEnd` methods to trim whitespace or some other set of characters from a string.

You use the `TrimStart` method to remove leading whitespace (or characters of your choosing) from the beginning of a string. To remove whitespace, you pass `TrimStart` the `Nothing` keyword as its parameter. Whitespace consists of the blank space characters, tabs, line feeds, form feeds, and carriage returns. You can also pass `TrimStart` an array of type `Char` in which case `TrimStart` trims the characters found in the array from the beginning of the string.

You use the `TrimEnd` method to remove trailing whitespace (or characters of your choosing) from a string. Like `TrimStart`, you can pass `TrimEnd` either `Nothing` or an array of `Char`.

The `Trim` method combines the behavior of `TrimStart` and `TrimEnd`. `Trim` removes both leading and trailing whitespace (or characters of your choosing) from a string. Unlike `TrimStart` and `TrimEnd`, there's no need to pass `Trim` the `Nothing` keyword to remove whitespace. If you call `Trim` without any parameter, it will remove whitespace. Like `TrimStart` and `TrimEnd`, however, you can pass `Trim` an array of type `Char` in which case `Trim` removes the characters found in the array from the start and end of the string.

The code in Listing 5.4, from `Trim.aspx`, demonstrates the use of the trim methods.

Listing 5.4 TRIM.ASPX—Experimenting with the Trim Methods

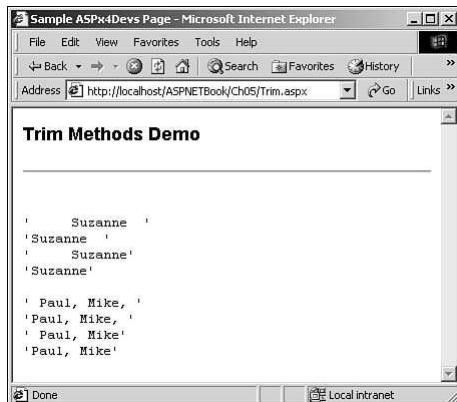
```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim Name As String = " Suzanne "
    Dim NameList As String = " Paul, Mike, "
    Dim TrimChars() As Char = {" ", "c", ",","c"}

    output.Text &= "<pre><p>"
    output.Text &= "<br />" & Name & ""
    output.Text &= "<br />" & Name.TrimStart(Nothing) & ""
    output.Text &= "<br />" & Name.TrimEnd(Nothing) & ""
    output.Text &= "<br />" & Name.Trim() & ""
    output.Text &= "</p><p>"
    output.Text &= "<br />" & NameList & ""
    output.Text &= "<br />" & NameList.TrimStart(TrimChars) & ""
    output.Text &= "<br />" & NameList.TrimEnd(TrimChars) & ""
    output.Text &= "<br />" & NameList.Trim(TrimChars) & ""
    output.Text &= "</p></pre>"
End Sub
```

The output produced by `Trim.aspx` is shown in Figure 5.3.

TIP

You can use the static `Char.IsWhiteSpace` method to determine if a character is considered whitespace by the .NET Framework.

**Figure 5.3**

This page demonstrates the use of the trim methods.

Comparing Strings

The `System.String` class provides three methods for comparing strings: the static `Compare` and `CompareOrdinal` methods and the instance method, `CompareTo`. All of the compare methods return a 0 if the strings are equal, a negative number if the first string is less than the second string, or a positive number if the first string is greater than the second string.

You can use the `Compare` method to compare two strings in a variety of ways. By default, `Compare` uses a case-sensitive comparison. For example:

```
Dim PhraseA As String = "ASP.NET for Developers"
Dim PhraseB As String = "ASP.NET for Developers"
Dim PhraseC As String = "asp.net for developers"

Result = String.Compare(PhraseA, PhraseB) '=0
Result = String.Compare(PhraseA, PhraseC) '=1
```

Optionally, you can pass `Compare` a third parameter indicating `True` to ignore case or `False` to regard case. For example (using the same strings):

```
Result = String.Compare(PhraseA, PhraseC, False) '=1
Result = String.Compare(PhraseA, PhraseC, True) '=0
```

You can also pass `Compare` a fourth parameter indicating which locale to use when making the comparison. This fourth parameter must be of the form of a `CultureInfo` object, which provides locale-specific information that may affect the comparison. For example, the following example compares `PhraseA` and `PhraseC`, disregarding case, and using "Polish (Poland)" locale information:

```
Result = String.Compare(PhraseA, PhraseC, True, _
New System.Globalization.CultureInfo("pl-PL")) '=0
```

You can also use `Compare` to compare a portion of a string to a portion of another string. In this case, you pass `Compare` the first string, the starting position in the first string, the second string, the starting position in the second string, and the length of the substring to compare. For example:

```
Result = String.Compare(PhraseA, 0, PhraseC, 0, 7) '=1
```

When comparing substrings, you can also pass sixth and seventh parameters to indicate whether to consider case and locale information, respectively.

You can use the `CompareOrdinal` method to compare two strings without considering the local national language or culture. Comparisons made with `CompareOrdinal` are case-sensitive, however. You can use `CompareOrdinal` to compare two whole strings or two substrings. For example:

```
Result = String.CompareOrdinal(PhraseA, PhraseC)           '-=-32
```

```
Result = String.CompareOrdinal(PhraseA, 8, PhraseC, 8, 3) '=0
```

Finally, you can use the `CompareTo` method to compare an instance of a string with another string. Comparisons made with `CompareTo` are case-sensitive. For example:

```
Result = PhraseA.CompareTo(PhraseC) '=1
```

Changing the Case of a String

You can use the `ToLower` and `ToUpper` methods to change the case of a string to lowercase or uppercase, respectively. For example:

```
Dim PhraseA As String = "ASP.NET for Developers"
Result1 = PhraseA.ToLower() '=asp.net for developers
Result2 = PhraseA.ToUpper() '=ASP.NET FOR DEVELOPERS
```

DateTime Arithmetic

The `System.DateTime` class provides functionality for the manipulation of `DateTime` values.

Now and Today

You can use the static properties `Now` and `Today` to retrieve the current date and time and the current date, respectively. For example:

```
Result = DateTime.Now    ' e.g., 7/27/2001 4:00:18 PM
```

```
Result = DateTime.Today ' e.g., 7/27/2001
```

You can also use the static `UtcNow` property to return the current date and time expressed as the Coordinated Universal Time (UTC; also known as Greenwich Mean Time or GMT).

Creating DateTime Values

You can create a `DateTime` value simply by declaring a variable of type `DateTime` and setting it to a `DateTime` constant. You use the # symbol to delimit `DateTime` constants. For example:

```
Dim BDay As DateTime = #6/25/2001#
Dim Appointment As DateTime = #12:30#
Dim DropOffCar As DateTime = #8/13/2001 08:30#
```

To convert a `String` value into a `DateTime` value, you can use the static method `Parse`. For example:

```
Dim Register As String = "5/14/2001"
Dim RegisterDate As DateTime
RegisterDate = DateTime.Parse(Register)
```

Comparing Dates

You can use the static `Compare` method to compare two `DateTime` values. `Compare` returns 0 if the `DateTime` values are equal, -1 if the first `DateTime` value is less than the second `DateTime` value, or 1 if the first `DateTime` value is greater than the second `DateTime` value.

For example:

```
Dim Date1 As DateTime = #6/25/2001#
Dim Date2 As DateTime = #6/26/2001#
Result = DateTime.Compare(Date1, Date2) '= -1
```

TIP

You can also use the static `Equals` and instance `Equals` methods to determine if two `DateTime` values are equal. These methods return `True` if the `DateTime` values are equal, or `False` if they are not.

Adding and Subtracting DateTime Values

The `DateTime` class provides a number of methods for adding and subtracting `DateTime` values: `AddYears`, `AddMonths`, `AddDays`, `AddHours`, `AddMinutes`, `AddSeconds`, `AddMilliseconds`, and `AddTicks`. These methods are illustrated by the code in Listing 5.5, from `DateAdd.aspx`.

Listing 5.5 DATEADD.ASPX—Adding and Subtracting DateTime Values

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim TheDate As DateTime = #6/25/2001 12:00#
    output.Text &= "<table border=""1"">" 
    output.Text &= "<tr><th>Original DateTime</th>" & _
```

Listing 5.5 continued

```

    "<th>Method</th><th>Resulting DateTime</th></tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddYears(-1)</td><td>" & TheDate.AddYears(-1) & "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddMonths(6)</td><td>" & TheDate.AddMonths(6) & "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddDays(45)</td><td>" & TheDate.AddDays(45) & "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddMinutes(-30)</td><td>" & TheDate.AddMinutes(-30) & _  

    "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddSeconds(90)</td><td>" & TheDate.AddSeconds(90) & _  

    "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddMilliseconds(1000)</td><td>" & _  

    TheDate.AddMilliseconds(1000) & "</td><tr>"  

output.Text &= "<tr><td>" & TheDate & "</td>" & _  

    "<td>TheDate.AddTicks(10000000)</td><td>" & _  

    TheDate.AddTicks(10000000) & "</td><tr>"  

    output.Text &= "</table/>"  

End Sub

```

The output generated by DateAdd.aspx is shown in Figure 5.4.

Original DateTime	Method	Resulting DateTime
6/25/2001 12:00:00 PM	TheDate.AddYears(-1)	6/25/2000 12:00:00 PM
6/25/2001 12:00:00 PM	TheDate.AddMonths(6)	12/25/2001 12:00:00 PM
6/25/2001 12:00:00 PM	TheDate.AddDays(45)	8/9/2001 12:00:00 PM
6/25/2001 12:00:00 PM	TheDate.AddMinutes(-30)	6/25/2001 11:30:00 AM
6/25/2001 12:00:00 PM	TheDate.AddSeconds(90)	6/25/2001 12:01:30 PM
6/25/2001 12:00:00 PM	TheDate.AddMilliseconds(1000)	6/25/2001 12:00:01 PM
6/25/2001 12:00:00 PM	TheDate.AddTicks(10000000)	6/25/2001 12:00:00 PM

Figure 5.4

Using the Add methods to add and subtract DateTime values.

You can use the Subtract method to subtract one DateTime value from another, producing a TimeSpan value. A TimeSpan value represents a period of time. You can also add a TimeSpan value to a DateTime value, returning a second DateTime value using the Add method.

Listing 5.6, from DateSubtract.aspx, illustrates the use of the Subtract and Add methods.

Listing 5.6 DATESUBTRACT.ASPX—The Add and Subtract Methods Manipulate TimeSpan Values

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim Date1 As DateTime = #6/25/2001#
    Dim Date2 As DateTime = #5/15/2001#

    Dim TSpan As TimeSpan = Date1.Subtract(Date2)
    Dim Date3 As DateTime = Date2.Add(TSpan)

    output.Text &= "<br />Date1 = " & Date1
    output.Text &= "<br />Date2 = " & Date2
    output.Text &= "<br />TSpan = Date1.Subtract(Date2) = " & _
        TSpan.ToString()
    output.Text &= "<br />Date2.Add(TSpan) = " & Date2.Add(TSpan)
End Sub
```

Unlike DateTime and numeric values, VB won't automatically coerce a TimeSpan into a String. However, you can use the ToString method to coerce a TimeSpan into a String as was done in Listing 5.6. The output from DateSubtract.aspx is shown in Figure 5.5.

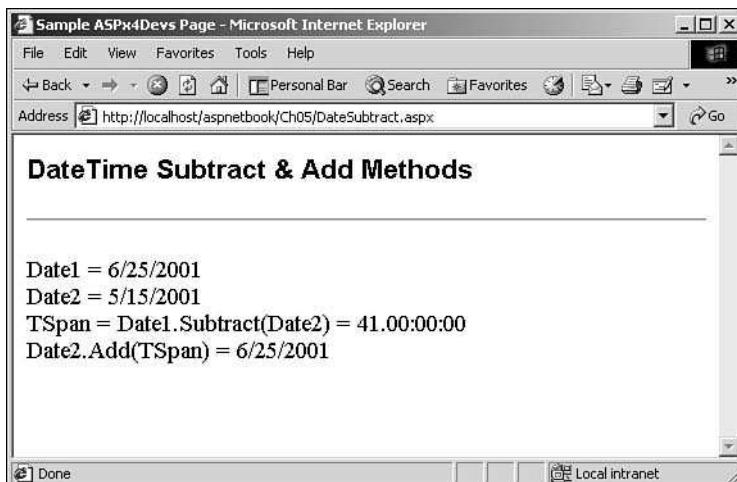


Figure 5.5

Using the Subtract and Add methods.

Parsing DateTime Values

The `DateTime` class provides a number of properties that you can use to retrieve parts of a `DateTime` value. You can use the `Year`, `Month`, and `Day` properties to retrieve the year, month, and day portions of a `DateTime` value, respectively. You can also use the `DayOfWeek` and `DayOfYear` properties to retrieve the day of week and day of year of a `DateTime` value, respectively. You can use the `Hour`, `Minute`, `Second`, `Millisecond`, and `Ticks` properties to retrieve the hour, minutes, seconds, milliseconds, and ticks portions of a `DateTime` value, respectively. A tick represents 100 nanoseconds. Finally, you can use the `Date` and `TimeOfDay` properties to retrieve the date portion and the time portion of a `DateTime` value.

Listing 5.7, from `DateTimeParse.aspx`, demonstrates the use of the `DateTime` parsing properties.

Listing 5.7 DATETIMEPARSE.ASPX—Parsing DateTime Values

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim TheDate As DateTime = #7/04/2001 09:03:45#
    output.Text &= "<table border=""1"">" &
    output.Text &= "<tr><th>Original DateTime</th>" & _
        "<th>Property</th><th>Returns</th></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Year</td><td>" & TheDate.Year & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Month</td><td>" & TheDate.Month & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Day</td><td>" & TheDate.Day & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.DayOfWeek</td><td>" & TheDate.DayofWeek & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.DayOfYear</td><td>" & TheDate.DayOfYear & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Hour</td><td>" & TheDate.Hour & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Minute</td><td>" & TheDate.Minute & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Second</td><td>" & TheDate.Second & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Millisecond</td><td>" & TheDate.Millisecond & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Ticks</td><td>" & TheDate.Ticks & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.Date</td><td>" & TheDate.Date & "</td></tr>" &
    output.Text &= "<tr><td>" & TheDate & "</td>" & _
        "<td>TheDate.TimeOfDay</td><td>" & _
        TheDate.TimeOfDay.ToString() & "</td></tr>" &
    output.Text &= "</table/>"
```

End Sub

The output generated by the code from `DateTimeParse.aspx` is shown in Figure 5.6. The `TimeOfDay` property returns a `TimeSpan` value.

A screenshot of a Microsoft Internet Explorer window titled "Sample ASPx4Devs Page - Microsoft Internet Explorer". The address bar shows the URL "http://localhost/ASPNETBook/Ch05/DateTimeParse.aspx". The main content area has a title "Parsing DateTime Values" and displays a table with 11 rows. The table has three columns: "Original DateTime", "Property", and "Returns". The data in the table is as follows:

Original DateTime	Property	Returns
7/4/2001 9:03:45 AM	TheDate.Year	2001
7/4/2001 9:03:45 AM	TheDate.Month	7
7/4/2001 9:03:45 AM	TheDate.Day	4
7/4/2001 9:03:45 AM	TheDate.DayOfWeek	3
7/4/2001 9:03:45 AM	TheDate.DayOfYear	185
7/4/2001 9:03:45 AM	TheDate.Hour	9
7/4/2001 9:03:45 AM	TheDate.Minute	3
7/4/2001 9:03:45 AM	TheDate.Second	45
7/4/2001 9:03:45 AM	TheDate.Millisecond	0
7/4/2001 9:03:45 AM	TheDate.Ticks	63129834225000000
7/4/2001 9:03:45 AM	TheDate.Date	7/4/2001
7/4/2001 9:03:45 AM	TheDate.TimeOfDay	09:03:45

Figure 5.6

The `DateTime` class provides a number of properties for parsing `DateTime` values.

Converting Values

The `System` classes support a variety of methods for converting values from one data type to another. You can use static methods of the `Convert` class to convert data. For example:

```
Dim Age As Integer = 42
Dim BDay As DateTime = #04/26/1958#
Dim Adult As Boolean = True
Dim AgeDbl As Double
Dim BDayStr As String
Dim AdultInt As Integer

AgeDbl = Convert.ToDouble(Age)
BDayStr = Convert.ToString(BDay)
AdultInt = Convert.ToInt32(Adult)
```

Some conversions that the .NET Framework considers illegal will throw exceptions. For example, an exception will be thrown if you attempt to convert a `DateTime` value into an `Int32` value.

You can use the `Parse` method of the numeric data types to convert `String` values into numbers. For example, to convert the string “1.00005” into a `Double`, you might use code like this:

```
Dim Dbl1 As Double
Dbl1 = Double.Parse("1.00005")
```

If the string representation of the number contains currency symbols, thousands separators, and other symbols, you can pass the `Parse` method an optional `NumberStyle` parameter. This parameter takes its values from the `NumberStyles` enumeration of the `System.Globalization` class. The `NumberStyles` enumeration values are summarized in Table 5.1

Table 5.1 NumberStyles Enumerations

Member	Description
<code>AllowCurrencySymbol</code>	Allows currency symbols.
<code>AllowDecimalPoint</code>	Allows decimal point characters.
<code>AllowExponent</code>	Allows exponents.
<code>AllowHexSpecifier</code>	Allows hexadecimal values.
<code>AllowLeadingSign</code>	Allows a leading sign character.
<code>AllowLeadingWhite</code>	Allows leading whitespace.
<code>AllowParentheses</code>	Allows parentheses.
<code>AllowThousands</code>	Allows thousands separator characters.
<code>AllowTrailingSign</code>	Allows a trailing sign character.
<code>AllowTrailingWhite</code>	Allows trailing whitespace.
<code>Any</code>	Allows any of the <code>AllowXXX</code> styles.
<code>Currency</code>	Allows all <code>AllowXXX</code> styles except for <code>AllowExponent</code> .
<code>Float</code>	Allows <code>AllowLeadingWhite</code> , <code>AllowTrailingWhite</code> , <code>AllowLeadingSign</code> , <code>AllowDecimalPoint</code> , and <code>AllowExponent</code> styles.
<code>HexNumber</code>	Allows <code>AllowLeadingWhite</code> , <code>AllowTrailingWhite</code> , and <code>AllowHexSpecifier</code> styles.
<code>Integer</code>	Allows <code>AllowLeadingWhite</code> , <code>AllowTrailingWhite</code> , and <code>AllowLeadingSign</code> styles.
<code>None</code>	Allows none of the <code>AllowXXX</code> styles.
<code>Number</code>	Allows <code>AllowLeadingWhite</code> , <code>AllowTrailingWhite</code> , <code>AllowLeadingSign</code> , <code>AllowTrailingSign</code> , <code>AllowDecimalPoint</code> , and <code>AllowThousands</code> styles.

NOTE

You will have to import the `System.Globalization` class to make use of the `NumberStyles` enumerations.

The following code illustrates the use of the NumberStyles styles:

```
Dim Db12, Db13 As Double  
Db12 = Double.Parse("$42", NumberStyles.AllowCurrencySymbol)  
Db13 = Double.Parse("$3,899.02", NumberStyles.Currency)
```

If you wish to allow for more than one of the NumberStyles styles, you can combine them together using the Or operator. For example:

```
Dim Db14 As Double  
Db14 = Double.Parse("$3,899.02", NumberStyles.AllowCurrencySymbol _  
    Or NumberStyles.AllowThousands Or NumberStyles.AllowDecimalPoint)
```



TIP

You can pass the Parse methods an optional third parameter to specify locale-specific information to be used for parsing the string.

Formatting Values

You can use the ToString method of the numeric and DateTime data types to format numbers and dates for output.

Formatting Numbers

You can use the ToString method of the numeric datatypes to format numbers using a standard formatting string or a custom formatting string. The standard formatting string characters are summarized in Table 5.2.

Table 5.2 Standard Numeric Format Characters

Format Character	Description
C or c	Currency format.
D or d	Decimal format.
E or e	Scientific (exponential) format; the case determines the case of "e" used.
F or f	Fixed-point decimal format.
G or g	General format.
N or n	Number.
R or r	Roundtrip format; ensures that numbers converted to strings will have the same value when they are converted back to numbers.
X or x	Hexadecimal; the case determines the case of hexadecimal characters.

You can also use custom format characters to create a custom format string. The custom format characters are summarized in Table 5.3.

Table 5.3 Custom Numeric Format Characters

Format Character	Description
0	Add a digit if one exists or 0 otherwise.
#	Add a digit if one exists or nothing otherwise.
,	Add a thousands separator character.
.	Add a decimal point character.
%	Add a percentage character and divide the value by 100.
E or e	Format the number using scientific notation.
\c or {c	Add one literal character where c is any character.
'characters' or "characters"	Add several literal characters.

The code in Listing 5.8, from `FormatNumber.aspx`, includes several examples of using the standard and custom numeric formatting strings with the `ToString` methods.

Listing 5.8 FORMATNUMBER.ASPX—Examples of Using the ToString Method and Various Standard and Custom Formatting Strings

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim dbl1 As Double = 1345.045
    Dim dbl2 As Double = 455356000
    Dim int1 As Int32 = 1345

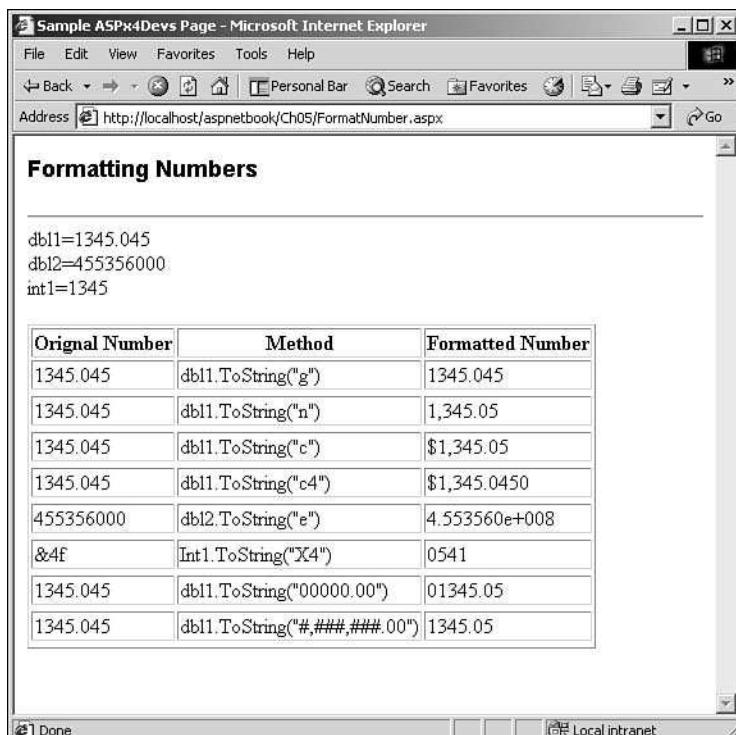
    output.Text &= "dbl1=" & dbl1 & "<br />"
    output.Text &= "dbl2=" & dbl2 & "<br />"
    output.Text &= "int1=" & int1 & "<br />"
    output.Text &= "<br />

    output.Text &= "<table border=""1"">
        output.Text &= "<tr><th>Original Number</th>" &
            "<th>Method</th><th>Formatted Number</th></tr>" 
        output.Text &= "<tr><td>" & "1345.045" & "</td>" &
            "<td>dbl1.ToString("g")</td><td>" &
            dbl1.ToString("g") & "</td><tr>" 
        output.Text &= "<tr><td>" & "1345.045" & "</td>" &
            "<td>dbl1.ToString("n")</td><td>" &
            dbl1.ToString("n") & "</td><tr>" 
        output.Text &= "<tr><td>" & "1345.045" & "</td>" &
            "<td>dbl1.ToString("c")</td><td>" &
            dbl1.ToString("c") & "</td><tr>" 
        output.Text &= "<tr><td>" & "1345.045" & "</td>" &
            "<td>dbl1.ToString("c4")</td><td>" &
            dbl1.ToString("c4") & "</td><tr>" 
        output.Text &= "<tr><td>" & "455356000" & "</td>" &
            "<td>dbl2.ToString("e")</td><td>" &
            dbl2.ToString("e") & "</td><tr>" 
        output.Text &= "<tr><td>" & "&#4f" & "</td>" &
            "<td>Int1.ToString("X4")</td><td>" &
```

Listing 5.8 continued

```
Int1.ToString("X4") & "</td><tr>"  
  
output.Text &= "<tr><td>" & "1345.045" & "</td>" & _  
"<td>dbl1.ToString("00000.00")</td><td>" & _  
dbl1.ToString("00000.00") & "</td><tr>"  
output.Text &= "<tr><td>" & "1345.045" & "</td>" & _  
"<td>dbl1.ToString("#,###,###.00")</td><td>" & _  
dbl1.ToString("#,###,###.00") & "</td><tr>"  
  
output.Text &= "</table/>"  
End Sub
```

The output from FormatNumber.aspx is shown in Figure 5.7. The last two entries in the table illustrate the use of custom formatting strings.

**Figure 5.7**

Examples of using the ToString methods to format numbers using various standard and custom formatting strings.

Formatting Dates

You can use the `ToString` method of the `DateTime` datatype to format a `DateTime` value for display. As when formatting numbers, you can use a standard `DateTime` formatting string or a custom `DateTime` formatting string.

The standard `DateTime` formatting strings are summarized in Table 5.4. Note that the formatting strings are case-sensitive.

Table 5.4 Standard DateTime Formatting Strings

Format Character	Date Format Pattern
d	MM/dd/yyyy
D	dddd, MMMM dd, yyyy
f	dddd, MMMM dd, yyyy HH:mm
F	dddd, MMMM dd, yyyy HH:mm:ss
g	MM/dd/yyyy HH:mm
G	MM/dd/yyyy HH:mm:ss
m or M	MMMM dd
r or R	ddd, dd MMM yyyy HH:mm:ss GMT
s	yyy-MM-ddTHH:mm:ss
t	HH:mm
T	HH:mm:ss
u	yyy-MM-dd HH:mm:ssZ
U	dddd, dd MMMM, yyyy HH:mm:ss
y or Y	yyy MMMM

The custom `DateTime` formatting strings are summarized in Table 5.5. Note that the formatting strings are case-sensitive.

Table 5.5 Custom DateTime Formatting Strings

Format Characters	Description
d	Day of the month
dd	Day of the month with leading zero
ddd	Three letter abbreviated name of the day of the week
dddd	Full name of the day of the week
M	Numeric month
MM	Numeric month with leading zero
MMM	Three letter abbreviated name of the month
MMMM	Full name of the month
y	Year without the century
yy	Year without the century, with leading zero
yyyy	Year including the century in four digits
gg	Period or era
h	Hour in 12-hour format
hh	Hour in 12-hour format with leading zero

Table 5.5 continued

Format Characters	Description
H	Hour in 24-hour format
HH	Hour in 24-hour format with leading zero
m	Minute
mm	Minute with leading zero
s	Second
ss	Second with leading zero
t	a for AM, p for PM
tt	AM/PM
z	Timezone offset (hour only)
zz	Timezone offset (hour only) with leading zero
zzz	Full timezone offset (hour and minutes) with leading zeros
:	Default time separator
/	Default date separator
% c	Standard format character where c is any standard format character (displays the standard format pattern associated with the format character)
\c	Literal character where c is any character

Listing 5.9, from `FormatDateTime.aspx`, demonstrates the use of the `ToString` method with a variety of standard and custom `DateTime` formatting strings.

Listing 5.9 FORMATDATETIME.ASPX—Formatting DateTime Values Using Various Standard and Custom DateTime Formats

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim SeattleEquake As DateTime = #02/28/2001 10:54#
    output.Text &= "<table border=""1"">
        <tr><th>Original DateTime</th>" & _
        "<th>Method</th><th>Formatted DateTime</th></tr>
        <tr><td>" & SeattleEquake & "</td>" & _
        "<td>SeattleEquake.ToString("d")</td><td>" & _
        SeattleEquake.ToString("d") & "</td><tr>
        <tr><td>" & SeattleEquake & "</td>" & _
        "<td>SeattleEquake.ToString("D")</td><td>" & _
        SeattleEquake.ToString("D") & "</td><tr>
        <tr><td>" & SeattleEquake & "</td>" & _
        "<td>SeattleEquake.ToString("s")</td><td>" & _
        SeattleEquake.ToString("s") & "</td><tr>
        <tr><td>" & SeattleEquake & "</td>" & _
        "<td>SeattleEquake.ToString("R")</td><td>" & _
        SeattleEquake.ToString("R") & "</td><tr>
        <tr><td>" & SeattleEquake & "</td>" & _
        "<td>SeattleEquake.ToString("u")</td><td>" & _
```

Listing 5.9 continued

```

SeattleEquake.ToString("u") & "
```

```

output.Text &= "<tr><td>" & SeattleEquake & "</td>" & _
"<td>SeattleEquake.ToString("")</td><td>" & _
SeattleEquake.ToString("U") & "</td><tr>"
```

```

output.Text &= "<tr><td>" & SeattleEquake & "</td>" & _
"<td>SeattleEquake.ToString("y")</td><td>" & _
SeattleEquake.ToString("y") & "</td><tr>"
```

```

output.Text &= "<tr><td>" & SeattleEquake & "</td>" & _
"<td>SeattleEquake.ToString("MMM\ -dd\ -yyyy gg hh:mm")</td><td>" & _
SeattleEquake.ToString("MMM\ -dd\ -yyyy\, gg hh:mm tt") & "</td><tr>"
```

```

output.Text &= "</table/>"
```

End Sub

Original DateTime	Method	Formatted DateTime
2/28/2001 10:54:00 AM	SeattleEquake.ToString("d")	2/28/2001
2/28/2001 10:54:00 AM	SeattleEquake.ToString("D")	Wednesday, February 28, 2001
2/28/2001 10:54:00 AM	SeattleEquake.ToString("t")	2001-02-28T10:54:00
2/28/2001 10:54:00 AM	SeattleEquake.ToString("R")	Wed, 28 Feb 2001 10:54:00 GMT
2/28/2001 10:54:00 AM	SeattleEquake.ToString("u")	2001-02-28 10:54:00Z
2/28/2001 10:54:00 AM	SeattleEquake.ToString("U")	Wednesday, February 28, 2001 10:54:00 PM
2/28/2001 10:54:00 AM	SeattleEquake.ToString("y")	February, 2001
2/28/2001 10:54:00 AM	SeattleEquake.ToString("MMMd\ -dd\ -yyyy gg hh:min")	Feb-28-2001, A.D. 10:54 AM

The output generated by `FormatDateTime.aspx` is shown in Figure 5.8.

Figure 5.8

Examples of formatting `DateTime` values using various standard and custom formatting strings.

Managing Arrays

The `System.Array` class provides a number of properties and methods for the manipulation of arrays. Some of these members overlap legacy functionality found in the VB language.

Determining the Boundaries of an Array

You can use the `GetLowerBound` and `GetUpperBound` methods of the `System.Array` class to retrieve the lower and upper bounds of a specified dimension of an array. Both methods take a single parameter, the dimension of the array, expressed as a zero-based value.

The following example, from `Array.aspx`, illustrates the use of `GetLowerBound` and `GetUpperBound` to determine the boundaries of the 0th dimension of the `MyFriends` array:

```
Dim MyFriends() As String = {"Suzanne", "Geoff", "Anna", _
    "Mike", "Ken"}
Dim i As Integer

For i = MyFriends.GetLowerBound(0) To MyFriends.GetUpperBound(0)
    output.Text &= "<br />" & MyFriends(i)
Next
```

TIP

Because all VB .NET arrays have a lower bound of 0, it's unlikely you'll have much need for the `GetLowerBound` method.

You can use the `Length` property to return the total number of elements of an array, across all of the array's dimensions. You can use the `Rank` property to return the total number of dimensions of the array.

For example, you could use the following code to determine the length and rank of the `MyFriends` array from the last example:

```
Result1 = MyFriends.Length '= 5
Result2 = MyFriends.Rank   '= 1
```

Sorting and Reversing the Elements of an Array

You can use the static `Sort` method to sort the elements of a one-dimensional array.

Listing 5.10, from `ArraySort.aspx`, illustrates the use of the `Sort` method to sort the `MyFriends` array.

Listing 5.10 ARRAYSORT.ASPX—Sorting an Array Using the Sort Method

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim MyFriends() As String = {"Suzanne", "Geoff", "Anna", _
        "Mike", "Ken"}
    Dim i As Integer
    Dim SearchPosition As Integer

    output.Text &= "The original array:"
    For i = MyFriends.GetLowerBound(0) To MyFriends.GetUpperBound(0)
        output.Text &= "<br />" & MyFriends(i)
    Next

    Array.Sort(MyFriends)
    output.Text &= "<br /><br />The array after using " & _
        "Array.Sort(MyFriends):"
    For i = MyFriends.GetLowerBound(0) To MyFriends.GetUpperBound(0)
```

Listing 5.10 continued

```

output.Text &= "<br />" & MyFriends(i)
    Next

    SearchPosition = Array.BinarySearch(MyFriends, "Mike")
    output.Text &= "<br /><br />" &
        "Position of 'Mike' element: " & SearchPosition
End Sub

```

The output from `ArraySort.aspx` is shown in Figure 5.9.

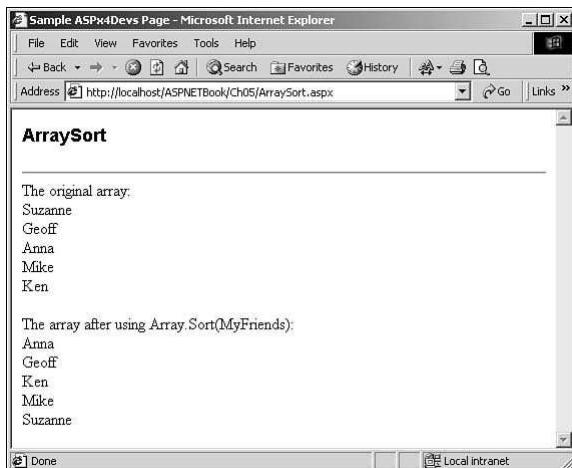


Figure 5.9

Sorting an array using the Sort method.

NOTE

By default, the `Sort` method sorts the whole one-dimensional array, but you can pass `Sort` two optional parameters: the element to start sorting at and the number of elements to sort.

You can also pass `Sort` an optional `IComparer` interface that it will use to govern the type of comparisons the sort algorithm uses. By default, the sort comparisons are case-sensitive.

You can use the static `Reverse` method to reverse the elements of a one-dimensional array. For example, you could use the following code to reverse the elements of the `MyFriends` array:

```
Array.Reverse(MyFriends)
```

Searching an Array

You can use the static `IndexOf` and `LastIndexOf` methods to search for an element in a one-dimensional array. Both `IndexOf` and `LastIndexOf` use a case-sensitive algorithm. `IndexOf` returns the element number (zero-based) of the first matching element, searching forward through the array. `LastIndexOf` returns the element number of the last matching element, searching backwards through the array. If no match is found, `IndexOf` and `LastIndexOf` return -1.

You can also use the static `BinarySearch` method to search for an element in a *sorted* one-dimensional array. `BinarySearch` returns the element number (zero-based) of the element containing the search item or a negative value (*not* necessarily -1) if no match was found. If there are multiple elements matching the search item, `BinarySearch` will return the position of one of the elements, but you have no guarantee which of the elements it will return. You need to use the `Sort` method prior to using the `BinarySearch` method or the result may be incorrect. By default, `BinarySearch` comparisons are case-sensitive but, like the `Sort` method, you can pass the `BinarySearch` method an optional `IComparer` interface that it will use to make the search comparisons.

Listing 5.11 (from `ArraySearch.aspx`) searches for the `Mike` element in the `MyFriends` array using `IndexOf`, `LastIndexOf`, and `BinarySearch`. Notice that there are two `Mike` elements in the array. The output from `ArraySearch.aspx` is shown in Figure 5.10.

Listing 5.11 ARRAYSEARCH.ASPX—Using the Various Array Methods to Search for an Item in an Array

```
Sub Page_Load(Src as Object, E as EventArgs)
    Dim MyFriends() As String = {"Suzanne", "Geoff", "Anna", _
        "Mike", "Ken", "Mike"}
    Dim i As Integer
    Dim SearchPosition As Integer

    output.Text &= "The original array:"
    For i = MyFriends.GetLowerBound(0) To MyFriends.GetUpperBound(0)
        output.Text &= "<br />" & MyFriends(i)
    Next
    SearchPosition = Array.IndexOf(MyFriends, "Mike")
    output.Text &= "<br /><br />IndexOf Position of 'Mike' element: " _
        & SearchPosition

    SearchPosition = Array.LastIndexOf(MyFriends, "Mike")
    output.Text &= "<br /><br />" & _
        "LastIndexOf Position of 'Mike' element: " _
        & SearchPosition

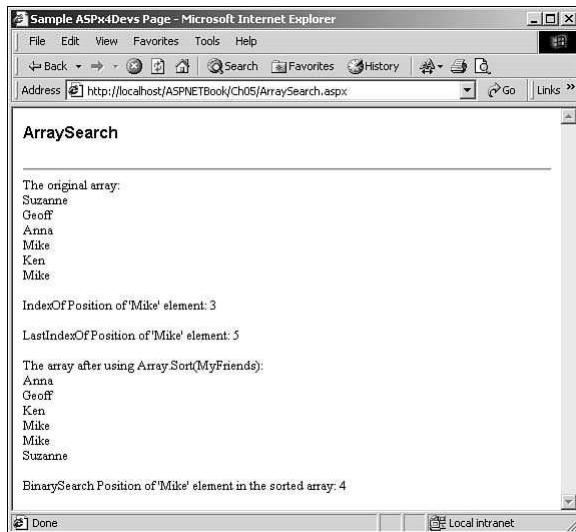
    Array.Sort(MyFriends)
    output.Text &= "<br /><br />" & _
        "The array after using Array.Sort(MyFriends):"
    For i = MyFriends.GetLowerBound(0) To MyFriends.GetUpperBound(0)
        output.Text &= "<br />" & MyFriends(i)
    Next
```

Listing 5.11 continued

```

SearchPosition = Array.BinarySearch(MyFriends, "Mike")
output.Text &= "<br /><br />" & _
    "BinarySearch Position of 'Mike' element " & _
    "in the sorted array: " & SearchPosition
End Sub

```

**Figure 5.10**

Searching for an element in an array.

NOTE

By default, `IndexOf`, `LastIndexOf`, and `BinarySearch` search the whole one-dimensional array, but you can pass the methods two optional parameters: the element to start searching at and the number of elements to search.

Summary

In this chapter you explored using the .NET Framework's System classes to make mathematical calculations, to manipulate strings, to work with `DateTime` values, to convert and format values, and to manage arrays.

Bear in mind, however, that this chapter has only scratched the surface of the functionality of the .NET Framework classes. More than likely, you'll need to use some piece of functionality that wasn't discussed here. In that case, your best resource is the .NET Framework SDK Documentation Help file that contains the complete reference to the System classes.

CHAPTER 6

Using Namespaces and Assemblies with Visual Basic .NET

In this chapter, you learn about namespaces and assemblies. You learn what assemblies are and where they fit into the Microsoft .NET framework. You also learn what a namespace is, how namespaces relate to assemblies, and how to create your own namespaces. Finally, you will learn how to compile and import assemblies into your own applications.

Namespaces and Assemblies

Object-oriented programming introduces the concept of creating classes to represent objects. Classes define all parts of an object, including its properties and methods. A class built using the Microsoft .NET framework is referred to as a managed class, or an *assembly*.

Assemblies (also sometimes referred to as namespace assemblies) are a fundamental unit of the Microsoft .NET framework. The entire framework consists of hundreds of assemblies that provide definitions to the objects available to the framework.

With hundreds of assemblies, finding the one you want could be a difficult task. To solve this issue, assemblies are organized into *namespaces*. Namespaces are a mechanism for grouping related objects together. All Microsoft .NET framework assemblies are contained in either the System or Microsoft namespaces.

For example, the Microsoft .NET framework contains a large number of assemblies that are related to data access. Namespaces can be used to logically group these related assemblies together. Figure 6.1 shows how the data access assemblies are organized in the Microsoft .NET framework.

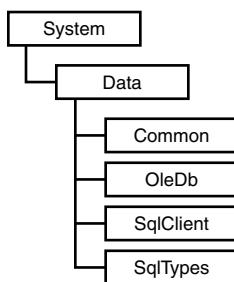


Figure 6.1

Assemblies organized by namespace.

As you will notice in Figure 6.1, namespaces can contain other namespaces. All of the Data assemblies are contained within the Data namespace, which is contained within the System namespace. Within the Data namespace, the Data assemblies are broken into four additional namespaces, Common, OleDb, SqlClient and SqlTypes. Breaking the data assemblies up into these namespaces helps to keep them more logically grouped and organized.

Relating Namespaces and DLL Assemblies

It is common to confuse the terms namespaces and assemblies, especially since they are often used together. It is important to remember the assemblies are the assemblage of classes. The namespaces merely provide a method to organize assemblies.

Namespaces have a many-to-many relationship with assemblies. That is, a namespace may span multiple assemblies. In addition, an assembly may be made up of classes in multiple namespaces.

Creating Assemblies

Creating an assembly consists of three fundamental steps: First, you need to define a namespace area that will contain your assembly. Next, you must create the assembly, including all of its properties, methods and events, within the namespace area. Finally, the namespace assembly must be compiled. The next few sections walk through these steps, beginning with defining a namespace area.

Defining a Namespace Area

You define a namespace area using Visual Basic .NET in the same way you would define a Function or Sub. You begin the namespace area by using the `Namespace` keyword followed by the name of the namespace. To define the end of the namespace area, add the `End Namespace` statement. All of the assemblies contained within the namespace must be defined within these two statements:

```
Namespace N1
```

```
...
```

```
End Namespace
```

In order to generate nested namespaces, which are namespaces that are contained within other namespaces, you can declare new namespace areas within another namespace area as shown below.

```
Namespace N1
```

```
    Namespace N2
```

```
        Namespace N3
```

```
        ...
```

```
    End Namespace
```

```
End Namespace
```

```
End Namespace
```

An optional way to declare nested namespaces is to declare the full namespace name, including parent namespaces, separated with the dot notation. The following shows this method.

```
Namespace N1.N2.N3
```

```
...
```

```
End Namespace
```

Creating an Assembly Within a Namespace Definition

Assemblies are made up of one or more compiled classes. When using the VB language, the source of an assembly will be a file with the VB extension. An example assembly containing a single class, `BasicMath`, is shown in Listing 6.1.

Listing 6.1 BASICMATH.VB—Creating an Assembly Within a Namespace Definition

```
Imports System
Namespace Math
    Public Class BasicMath
        Public Shared Function Add(X As Integer, Y As Integer)
            Return X + Y
        End Function
        Public Shared Function Subtract(X As Integer, Y As Integer)
            Return X - Y
        End Function
        Public Shared Function Multiply(X As Integer, Y As Integer)
            Return X * Y
        End Function
    End Class
End Namespace
```

The `Math` namespace contains a single class, `BasicMath`. The assembly name in this example is also named `BasicMath`.

NOTE

Classes, methods and properties were discussed in detail in Chapter 5, “Understanding Visual Basic.NET Syntax and Structure.”

A second example, the `Geometry` assembly, containing two classes, `Rectangle` and `Circle`, is shown in Listing 6.2.

Listing 6.2 GEOMETRY.VB—This Assembly Contains Two Classes

```
Namespace Shapes
    Public Class Rectangle
        Private HeightVal As Long
        Private WidthVal As Long
        Public Sub New(ByVal RectHeight As Long, ByVal RectWidth As Long)
            HeightVal = RectHeight
            WidthVal = RectWidth
        End Sub
```

Listing 6.2 continued

```
Public Property Height() As Long
    Get
        Return HeightVal
    End Get
    Set(ByVal Value As Long)
        HeightVal = Value
    End Set
End Property
Public Property Width() As Long
    Get
        Return WidthVal
    End Get
    Set(ByVal Value As Long)
        WidthVal = Value
    End Set
End Property
Public Function Area() As Long
    Return Height * Width
End Function
Public Function Perimeter() As Long
    Return ((Height * 2) + (Width * 2))
End Function
End Class
Public Class Circle
    Private RadiusVal As Long
    Public Sub New(ByVal Radius As Long)
        RadiusVal = Radius
    End Sub
    Public Property Radius() As Long
        Get
            Return RadiusVal
        End Get
        Set(ByVal Value As Long)
            RadiusVal = Value
        End Set
    End Property
    Public Function Area() As Long
        Return Radius^2*System.Math.PI
    End Function
    Public Function Circumference() As Long
        Return 2*Radius*System.Math.PI
    End Function
End Class
End Namespace
```

In this example, the Shapes namespace contains two classes, Rectangle and Circle, which live inside of the Geometry assembly.

Creating an Assembly Without a Namespace Definition

You can create assemblies without using the Namespace statement. In this case, the classes in the assembly become part of the application namespace that is global to the application.

Compiling an Assembly

Assemblies are compiled using the Visual Basic .NET Compiler, vbc.exe. This is included with the Microsoft .NET Framework SDK. The compiler supports a number of compile options that can be viewed by simply typing vbc at the command line. The following is the command used to compile the Geometry assembly:

```
vbc /t:library /out:bin\Geometry.dll Geometry.vb
```

NOTE

If you have installed the Microsoft .NET Framework via VS .NET, you may have to manually set the path to the VB. NET Command Line Compiler (vbc.exe).

This statement includes two required compile options. The first, /t:, indicates the target type. In this case, you are generating a library (or assembly). The second option, /out:, indicates the output of the compiler. In order for an assembly to be usable in your application, it must reside in the application's bin directory. So, in the case of the compile statement above, you are generating a DLL library named Geometry.dll in the bin directory.

NOTE

The compiler will not generate the bin directory for you. You must make sure the bin directory exists before compiling. Neglecting to do so will result in a compiler error.

In the compile statement above, after both of the compile options, you specify the source file(s) that are to be compiled. In this case you are compiling the file Geometry.vb.

Importing Assemblies

Assemblies can be imported into other assemblies or into your ASP.NET pages. To import them into other assemblies, you use the Imports statement, which should be inserted at the top of the assembly definition. The syntax for the Imports statement is

```
Imports [namespace]
```

Consider the Geometry.vb assembly, which resides in the Shapes namespace. In order to import the Geometry assembly into another assembly, you would use the following statement:

```
Imports Shapes
```

Assemblies can also be imported into ASP.NET pages. This is done with the `Import` directive. The syntax for the `Import` directive is

```
<% @Import namespace="[namespace]" %>
```

To import the Geometry assembly, you would insert the following `Import` directive at the top of the ASP.NET page:

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

NOTE

There's no need to import classes that were defined within assemblies created without a namespace definition. These classes are available throughout the application.

Using Imported Assemblies

Once you have imported the assembly into your application, it's ready to use. Assemblies are used just like any other assembly in the Microsoft .NET Framework. Listing 6.3 shows an example of using the Math.BasicMath assembly (shown earlier in Listing 6.1) in another assembly.

Listing 6.3 ADVANCEDMATH.VB—Creating an Assembly That References Another Assembly

```
Imports System
Imports Math
Namespace Math2
    Public Class AdvancedMath
        Public Shared Function Square(X As Integer)
            Return BasicMath.Multiply(X, X)
        End Function
    End Class
End Namespace
```

In this assembly, you execute the `Multiply` method of the `BasicMath` assembly. Notice that the first two lines of the assembly contain `Imports` statements.

Using an assembly from an ASP.NET page is virtually the same once the assembly has been imported. Listing 6.4 shows an example of using the `AdvancedMath` assembly defined in Listing 6.3 above.

Listing 6.4 BASICMATH.ASPX—Importing and Using Assemblies from ASP.NET

```
<%@ Page Description="document description" %>
<%@ Import namespace="Math2" %>
<script Language="VB" runat="server">
    sub SquareNumber(sender as object, e as EventArgs)
        Try
            If Not Number.Text = "" Then
                Number.Text = AdvancedMath.Square(Number.Text)
            End If
        Catch
            Number.Text = "Invalid Entry or Too Big"
        End Try
    end sub
</script>
<html>
<body>
<h2>Importing and Using assemblies from ASP.NET</h2>
<hr />
<form runat="server">
<table border="0">
    <tr>
        <td>
            Number to Square
        </td>
        <td>
            <asp:TextBox runat="server" id="Number" />
        </td>
    </tr>
    <tr>
        <td>
            &nbsp;
        </td>
        <td>
            <asp:Button Text="Square Number" OnClick="SquareNumber" runat="server" />
        </td>
    </tr>
</table>
</form>
</body>
</html>
```

Compiling with Imported Namespaces

In the previous section, you learned how to import a namespace assembly from another assembly. Assemblies that import other assemblies require another option when they are compiled. The other option is the reference option, /r, which references another assembly during the compile. For example, to compile the AdvancedMath assembly defined previously in Listing 6.3, you would use the following compile statement:

```
vbc /t:library /out:bin\AdvancedMath.dll AdvancedMath.vb /r:bin\BasicMath.dll
```

Notice the reference option at the end of the compile statement that references the BasicMath.dll in the bin directory of the application. This statement will generate an assembly library named AdvancedMath.dll.

Summary

Assemblies are a very important component in the Microsoft .NET framework. In this chapter, you learned what namespaces and assemblies are, and how they relate to each other. You learned how to create assemblies, compile them, and use them in other assemblies and in ASP.NET pages.

PART III

BUILDING WEB PAGES WITH ASP.NET

- 7 Understanding ASP.NET Web Forms
- 8 Creating Simple Web Pages with the HTML Server Controls
- 9 Creating Interactive Forms with Web Form Server Controls
- 10 Designing Advanced Interfaces with Web Form List Controls
- 11 Improving Your User Interfaces with Validation Controls
- 12 Adding User Controls to Your Web Forms

CHAPTER 7

Understanding ASP.NET Web Forms

One of the most radical changes in the programming model for Microsoft Web development is the introduction of the ASP.NET Web Forms model for creating Web pages. Unlike typical linear, stateless Web programming models, ASP.NET Web Forms provide an object-style, event-oriented, state-managed approach to creating robust and scalable Web pages. In addition, Microsoft has made it very easy to extend the existing Web Form programming model with new controls and even new forms of the Web Page object itself. This results in a very flexible and rich programming environment for Web applications.

In this chapter, you'll learn the details of the Web Forms code model, including alternate ways to organize your source code and the full life cycle of ASP.NET Web Forms. You'll also learn about the Web Form event cycle and how you can create Web form events and server-side code blocks to respond to client-side user events.

Finally, you'll learn how the ASP.NET Web Forms model provides state management within pages and how to take advantage of this state management model to increase the performance of your applications.

Understanding the Web Forms Code Model

The ASP.NET Web Forms code model is a new and unique way to approach programming for the Web. A number of features of Web Forms make them unique. First, they are created using server-side programming to handle client-side events. This means that Web Forms can run on virtually any browser base.

Also, since Web Forms are based on the .NET platform Common Language Runtime (CLR), they provide a true object-oriented programming model that includes full inheritance, type safety, and dynamic compilation. In addition, ASP.NET Web Forms can be programmed using any CLR-compliant language that supports just-in-time compilation, including Visual Basic, C#, Jscript .NET, and other third-party languages.

ASP.NET Web Forms also provide a base for hosting smart server-side controls that can participate fully in the page object life cycle and emit customized code based on the client browser requesting the page. This reduces the need for client-side browser coding and increases the “reach” of your Web application while reducing the amount of coding needed to complete a Web page. Finally, ASP.NET Web Forms offer a scalable page-level state management model that makes it much easier to create Web pages with a high degree of user interaction.

In-Page versus Code-Behind Format

Another important feature of ASP.NET Web Forms is that the coding model supports clear separation between HTML/XML markup and server-side code that animates the page. In the past it has been difficult to create quality Web pages without creating a kind of “spaghetti code” mixture of markup, client-side, and server-side code within the page. However, ASP.NET Web Forms offer a number of ways in which you can create Web pages that keep code and markup clearly separated.

In-Page Coding of ASP.NET Web Forms

One way in which you can easily separate server-code and markup is to take advantage of ASP.NET Web Forms’ ability to host server-side code within the actual ASPX page itself. Creating a script block at the top of the Web page usually does this. This single block of code can handle all the events and code processing needed for typical interactive Web pages. Listing 7.1 shows an example of this in-page coding style.

Listing 7.1 INPAGE.ASPX—Sample ASP.NET Web Form Using In-Page Coding Style

```
<%@ Page Description="in-page coding style" %>

<script LANGUAGE="vb" RUNAT="server">

sub Page_Load(sender as Object, e As EventArgs)

    if Page.IsPostBack=false then
        submit.Text="Click me"
        header.innerHTML="In-Page Coding Style Web Form"
    end if

end sub
```

Listing 7.1 continued

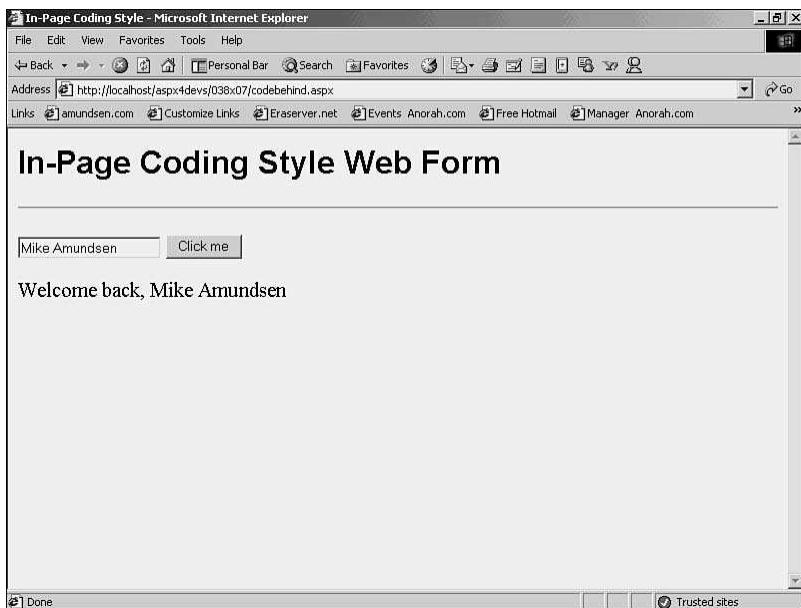
```
sub submit_click(sender as object, e as EventArgs)  
  
    postBack.Text="Welcome back, " & nameInput.Text  
  
end sub  
  
</script>  
  
<html>  
<head>  
    <title>In-Page Coding Style</title>  
</head>  
<body>  
  
    <h2 id="header" runat="server" />  
    <hr />  
  
    <form runat="server">  
        <asp:TextBox id="nameInput" runat="server" />  
        <asp:Button id="submit" OnClick="submit_click" runat="server" />  
    </form>  
  
    <asp:Label id="postBack" runat="server" />  
  
</body>  
</html>
```

Notice that the code in Listing 7.1 is all confined to a single `<script>` block on the page. ASP.NET Web Forms make it very easy to create code pages that encapsulate all the server-side execution code in a single place on the page. In this way, it is much easier to maintain the code for this page and much safer for HTML and graphics programmers to update the page without breaking execution code.

NOTE

The code example in Listing 7.1 may contain a number of keywords and markup elements that are unfamiliar to you. You'll learn the details of all ASP.NET Web Forms controls and coding methods later in Part 3—"Net Building Web Pages with ASP.NET." For now, it's important to recognize that ASP.NET Web Forms allow you to create pages that clearly separate code and markup.

Figure 7.1 shows an example of the code from Listing 7.1 running in Microsoft Internet Explorer.

**Figure 7.1**

Viewing the INPAGE.ASPX document in a browser.

Code-Behind Coding of ASP.NET Web Forms

Another very powerful way to create ASP.NET Web Forms is to use the *code-behind* format for programming Web pages. This method allows you to take all server-side code out of the markup page and place it in a completely separate file. This makes it even easier to maintain the markup safely without accidentally breaking server-side execution code.

In order to support the code-behind format, you need to first create two files: one to hold the markup information and another to hold the server-side execution code. The markup file (CODEBEHIND.ASPX) will have all the HTML/XML code plus a page directive as the very first line of the file. This directive will indicate that all server-side code can be found in another file in the Web site. Below is a typical page directive that accomplishes this task.

```
<%@ Page Inherits="CBPage" Src="CBPage.vb" %>
```

Next, you need to create a new file that contains all the server-side code needed to support the markup page. It is also important to note that some additional code is needed at the top of the page in order to support the code-behind model. This code is required in order to properly link both the code and markup pages and to be sure to define within the code page all scripted components that appear on the markup page.

For example, the code block seen in Listing 7.1 could be altered to support code-behind format by moving it to a new file (CBPAGE.VB) and adding the required header information seen in Listing 7.2.

Listing 7.2 CBPAGE.VB—Supporting Code-Behind Coding of ASP.NET Web Forms

```
Option Strict Off

Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
Imports System.Web.UI.HtmlControls

Public Class CBPage : Inherits Page

    Public nameInput As TextBox
    Public header As HtmlGenericControl
    Public submit As Button
    Public postBack as Label

    sub Page_Load(sender as Object, e As EventArgs)
        if Page.IsPostBack=false then
            submit.Text="Click me"
            header.innerHtml="In-Page Coding Style Web Form"
        end if
    end sub

    sub submit_click(sender as object, e as EventArgs)
        postBack.Text="Welcome back, " & nameInput.Text
    end sub

end Class
```

Along with a list of imported assembly references at the top of the page, notice the declaration of a public class (called CBPage) that inherits the Page base class. You then see the public declarations of each of the markup elements that are referenced by code within the class. With this page class completed, you only need to modify the markup page (now called CODEBEHIND.ASPX) by removing the <script> block and adding the proper page directive at the top of the file. Listing 7.3 shows an example of this page.

Listing 7.3 CODEBEHIND.ASPX—Updated Markup to Support Code-Behind Style

```
<%@ Page Inherits="CBPage" Src="CBPage.vb" %>

<html>
<head>
<title>In-Page Coding Style</title>
</head>
<body>

<h2 id="header" runat="server" />
<hr />

<form runat="server">
    <asp:TextBox id="nameInput" runat="server"/>
    <asp:Button id="submit" OnClick="submit_click" runat="server"/>
</form>

<asp:Label id="postBack" runat="server" />

</body>
</html>
```

As you can see, there is no execution code at all in the page that users request. The only link between the markup and execution code file is found in the page directive at the top of the ASPX document.

NOTE

If you are using Visual Studio .NET to create your pages, you'll discover that *all* ASP.NET Web Forms use the code-behind style. However, careful inspection of the VS .NET page directive will show that the format of this directive is slightly different.

It is not important to understand the differences, but it is vital that you not confuse the two formats. Using the wrong page directive format will result in compilation errors.

The Web Form Object Life Cycle

Along with the added power of in-page and code-behind programming, the ASP.NET Web Forms also offer a true object-oriented experience based on a Page base class. This class also publishes a number of initialization, rendering, and termination events that programmers can use to efficiently build and execute Web pages.

Unlike the more traditional Active Server Page static programming model, ASP.NET Web Forms are fully compiled class objects that produce HTML dynamically upon page requests from users. Also, unlike ASP solutions where page data is stored in text and interpreted each time, ASP.NET pages are compiled into DLL assemblies and are

executed directly from the compiled code. This allows for much more control by the programmer and a much more efficient execution on the server.

When an ASP.NET Web Forms page is requested, it goes through three distinct phases in its life cycle. These phases are

- Initialization
- Event-Handling (Rendering)
- Termination

In each of these phases, programmers can add custom code to create dynamic and efficient pages.

NOTE

Actually, there are a number of additional page-level events, but Web page programmers do not commonly use them. Instead, these added events are used by the various Web controls that need to handle their own initialization, data binding, and rendering within the page class itself.

The initialization task is handled in the `Page_Load` event. At this time, any input data sent from the client is evaluated and processed. If this is not the first time the page was displayed to the user, additional postback processing is done in order to update the page. Finally, any page-level state management is restored from the previous page call.

In the middle phase of the page life cycle—Rendering—the event handlers for all the controls on the page are executed. If validation controls appear on the page, the validation events also fire at this time. Finally, in the `Page_Unload` event, programmers can perform basic cleanup work such as closing files and databases along with the release of any objects accessed during the page load.

WARNING

It is important that expensive resources (like database connections) be explicitly closed. Otherwise, they will remain open until the next garbage collection occurs. On a heavily loaded server, too many open resources can exhaust memory.

Handling Client-Side Events on the Server

One of the more important features of the ASP.NET Web Forms model is its ability to support client-side events through server-side code. Literally, when a user clicks on a button on the client-side page, a server-side event is executed. This means that the page must be sent back to the server for processing each time the user fires off an event.

In order to support client-side events with server-side code, the control must have a `runat="server"` attribute. This attribute tells ASP.NET to create dynamic code that links the server-side code to the control. For example, the following code snippet shows how to inform ASP.NET that you wish to add server-side code support for the HTML text input control:

```
<input type="text" id="textInput" runat="server"/>
```

NOTE

While it is possible to create ASP.NET Web Forms using standard HTML controls that do not have the `runat="server"` tag, these traditional Web pages will not support server-side code for the events that fire on the client.

To handle this scenario, the Web Forms model supports *round-tripping* between the client and server. In round-tripping, each time the user presses a button, the page is sent to the server, the inputs are interpreted at the server, the page is rebuilt, and then this rebuilt page is sent back to the client again. This round-tripping continues whenever the user causes a client-side event to fire server-side code.

NOTE

Some ASP.NET Web Forms controls (notably the validation controls) are able to handle some events on the client-side in order to reduce round trips to the server.

Web Form Event Handling

Along with the ability to provide a self-posting form environment that allows pages to automatically round trip back to the server, ASP.NET Web Forms also allow programmers to define and respond to additional events on the page such as button clicks, list selection, and clicking on an anchor tag.

Even more important, ASP.NET Web Forms allow programmers to control how some page elements respond to clicks. This allows programmers to control the degree of interaction that the form provides. For example, list items (along with some other controls) can optionally cause a postback event directly or simply record the change and allow buttons on the page to cause the postback event.

In the next section, you'll learn how to define client-side events for server controls and how to respond to these events with the proper code blocks on the server.

Defining Web Form Control Events

The process of handling client-side events with server code involves completing just two tasks: defining control events and adding server-side code that responds to the event. First, you must define an event and associate it with a control element on the page. This will register the control to emit a block client-side event code. Depending on the browser, this will be done using either a snippet of client-side JavaScript or a standard HTML 3.2 POST FORM.

The following code snippet shows how to add a client-side event to a server control:

```
<asp:Button OnClick="button_click" runat="server" />
```

You can also register a server-side event for the standard HTML controls. You do this by using the `OnServerClick` attribute:

```
<input type="submit" OnServerClick="submit_click" runat="server" />
```

TIP

When working with standard HTML controls, you can define both a client-side and server-side event. Use the `OnClick` attribute for the client-side JavaScript event and the `OnServerclick` attribute for the server-side event.

Now that you know how to register page elements to participate in the ASP.NET Web Forms event model, you are ready to add the server-side code to respond to these events, as described in the next section.

Responding to Web Form Events

Once you add the event attribute to the page element, you need to add a matching server-side code block that will be executed when the client-side event is fired. All server-side events, or event handlers, have a similar signature or parameter list. The parameter list consists of two arguments. The following code snippet shows a typical declaration for a server-side event handler:

```
Sub myButton_Click(Sender as Object, Arg as EventArgs)
```

Inspecting the Object Argument of Web Forms Events

The first argument is of the type `Object` and represents the actual page element that fired the event. This argument can be used to determine which element on the page the user clicked. This way, you can add several event elements on the page and have them all fire off the same event on the server. By inspecting the object type and properties, you can determine which client-side object sent the message to the server. Listing 7.4 shows an example that takes advantage of this feature.

Listing 7.4 EVENTOBJECT.ASPX—Inspecting the Object Argument of Server-Side Event Handlers

```
<%@ Page Description="event objects" %>

<script LANGUAGE="vb" RUNAT="server">

sub Page_Load(sender as Object, e As EventArgs)

    if Page.IsPostBack=false then
        header.innerHTML="Event Objects"
    end if

end sub
```

Listing 7.4 continued

```
sub event_response(sender as object, e as EventArgs)

    dim objButton as Button = CType(sender, Button)
    postBack.Text = "You pressed: " & objButton.id

end sub

</script>

<html>

<head>
<title>Web Form Events</title>
</head>
<body>

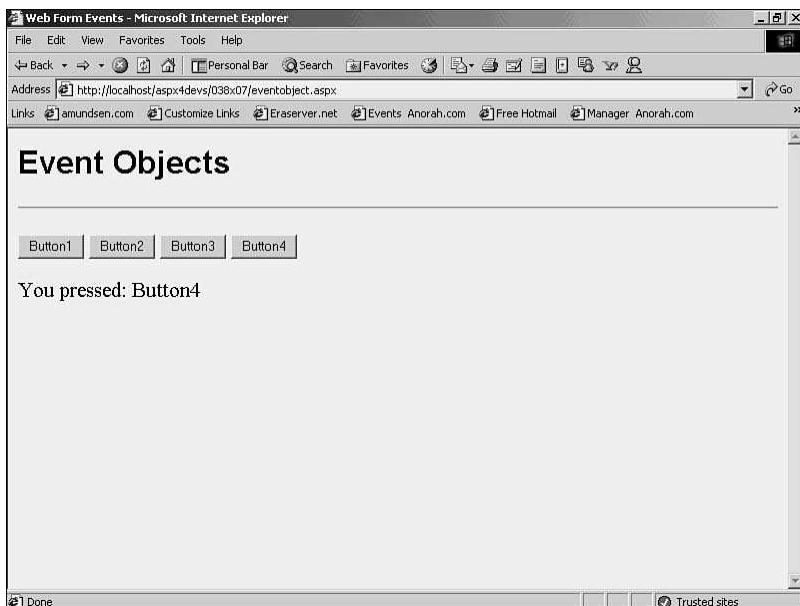
<h2 id="header" runat="server" />
<hr />

<form runat="server">
<asp:Button id="Button1" Text="Button1"
    OnClick="event_response" runat="server"/>
<asp:Button id="Button2" Text="Button2"
    OnClick="event_response" runat="server"/>
<asp:Button id="Button3" Text="Button3"
    OnClick="event_response" runat="server"/>
<asp:Button id="Button4" Text="Button4"
    OnClick="event_response" runat="server"/>
</form>

<asp:Label id="postBack" runat="server" />

</body>
</html>
```

When executing the preceding page, pressing each button will send a message to the server-side code that will report back to the button that was pressed on the client. Figure 7.2 shows how this looks in Microsoft Internet Explorer.

**Figure 7.2**

Accessing the Object argument of an ASP.NET event handler.

Accessing the Event Arguments

The second parameter of all ASP.NET Web Forms event handlers returns the event arguments. Most all page elements return an object of the type EventArgs. This is the base class for all event argument objects.

NOTE

Several of the more complex page elements (lists, grids, etc) have customized event argument objects that are derived from the EventArgs base class. You need to check the ASP.NET documentation to determine the exact object type needed to properly complete the event handler signature.

The code example in Listing 7.5 shows how to use the event arguments passed from a DataGrid page element. In this example, the DataGrid's sort event passes a custom version of the EventArgs base class called DataGridSortCommandEventEventArgs. This object has a property called SortExpression that can be used to enable sorting of the data in the grid.

Listing 7.5 EVENTARGS.ASPX—Handling the Custom EventArgs Object of the DataGrid Page Element

```
<%@ Page Description="handling event arguments" %>
<%@ Import Namespace = "System.Data" %>

<script language="vb" runat="server">

dim SortExpression as string

sub Page_Load(sender as object, args as EventArgs)

    if IsPostBack=false then
        if sortExpression="" then
            sortExpression="IntegerValue"
        end if

        dataGrid.DataSource = CreateDataSource()
        dataGrid.DataBind()
    end if

end sub

sub dataGrid_Sort(sender as object, args as dataGridSortCommandEventArgs)

    sortExpression = args.SortExpression.ToString()
    sortColumn.Text = "You clicked: " & sortExpression

    dataGrid.DataSource = CreateDataSource()
    dataGrid.DataBind()

end sub

function CreateDataSource() as ICollection

    dim Rand_num as Random = new Random()
    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim dv as DataView
    dim i as Integer

    with dt.Columns
        .Add(new DataColumn("IntegerValue", GetType(Int32)))
        .Add(new DataColumn("StringValue", GetType(string)))
        .Add(new DataColumn("CurrencyValue", GetType(double)))
    end with

    for i = 0 to 10 step 1
        dr = dt.NewRow()
        dr(0) = i
    next
end function
```

Listing 7.5 continued

```

dr(1) = "Item " & i.ToString()
dr(2) = 1.23 * Rand_Num.Next(1,15)
dt.Rows.Add(dr)
next

dv = new DataView(dt)
dv.Sort = sortExpression

return(dv)

end function

</script>

<html>
<body>
<h2>Handling Event Arguments</h2>
<hr />

<asp:Label id="sortColumn" runat="server" />

<form runat="server">
    <asp:DataGrid id="dataGrid" AllowSorting="true"
        OnSortCommand="dataGrid_Sort" runat="server" />
</form >

</body >
</html >

```

The `DataGrid_Sort` method in Listing 7.5 shows how you can access properties of the `EventArgs` object passed in each event handling method. The exact properties available for page elements differ depending on the page element in use. You need to check the ASP.NET documentation for each object in order to know just what type of `EventArgs` object is used and what properties of that object are available.

TIP

If you are using Visual Studio .NET to build your ASP.NET Web Forms, you will not need to construct your event handler signatures. Visual Studio .NET will automatically create the methods with the proper arguments for you.

Using the AutoPostBack Property

Along with common events such as a click event on buttons, some page elements support postback events for things like selecting items in a list or changing the contents of a text box. Since these kinds of “secondary” events can happen quite frequently on a form, these events do not automatically cause an event to fire on the server.

Instead, you need to add a special attribute to the server control in order to make sure it will create a client-side event that will send a message back to the server for processing. This special attribute is the AutoPostBack attribute. In order to force these controls to fire events on the server, you need to set the AutoPostBack attribute to true.

Listing 7.6 shows an example that uses the AutoPostBack attribute to register a server-side event for the DropDownList control.

Listing 7.6 AUTOPOSTBACKEVENTS.ASPX—Registering an AutoPostBack Event for a DropDownList Page Element

```
<%@ Page Description="postback events" %>

<script LANGUAGE="vb" RUNAT="server">

sub sportList_changed(sender as object, e as EventArgs)
    postBack.Text="You selected: " & sportList.selectedItem.Value
end sub

</script>

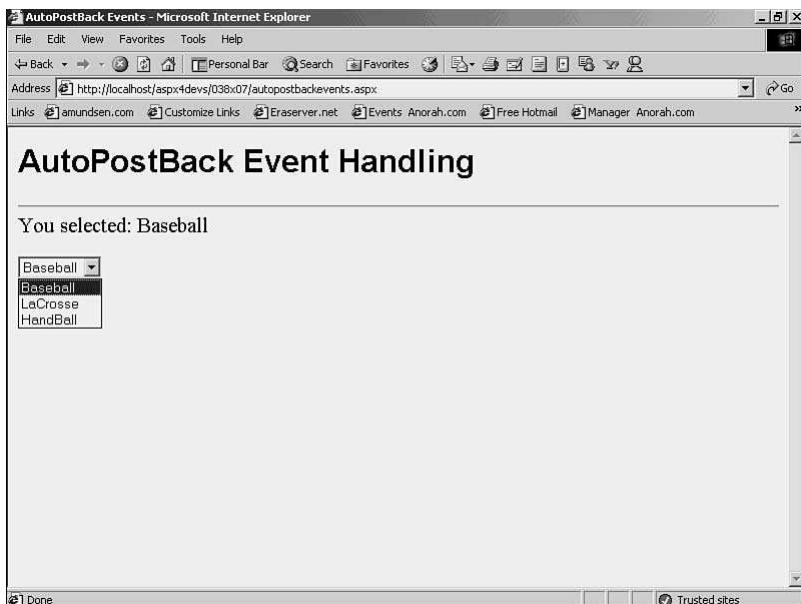
<html>
<body>
<h2>AutoPostBack Event Handling</h2>
<hr />

<asp:Label id="postBack" runat="server" />

<form runat="server">
    <asp:DropDownList id="sportList"
        OnSelectedIndexChanged="sportList_changed"
        AutoPostBack="True" runat="server">
        <asp:ListItem>Baseball</asp:ListItem>
        <asp:ListItem>LaCrosse</asp:ListItem>
        <asp:ListItem>HandBall</asp:ListItem>
    </asp:DropDownList>
</form>

</body>
</html>
```

When you load this page in a browser and select an item in the list, an event will fire on the server and display the resulting selection on the page (see Figure 7.3)

**Figure 7.3**

Testing the AutoPostBack event for ASP.NET Web Forms.

Automatic State Management with Web Forms

Along with the ability to provide server-side handling of client-side events, ASP.NET Web Forms also provide a very powerful state management model. This allows Web programmers to treat client-side page elements as local objects that can be easily programmed.

In addition to this improved programming experience, ASP.NET Web Forms also automatically save and restore the state of common page elements such as text boxes, lists, and other user input elements. This greatly improves the functionality of Web forms and also reduces the amount of coding needed to create user-friendly forms.

The easiest way to handle state management for ASP.NET Web Forms is to use automatic view state management. This is a feature built into the ASP.NET Web Forms object model and into the controls that appear on the page.

Most controls support the `EnableViewState` attribute. By default, this attribute is set to true. When set to true, the contents (along with some additional information) of the input elements are cached in a hidden input on the HTML page. Each time the page is sent back to the user, this cache is updated. This allows the page to automatically maintain the state of text boxes, radio buttons, lists, grids, and other elements.

Listing 7.7 shows how automatic view state management works.

Listing 7.7 AUTOMATICVIEWSTATE.ASPX—Illustrating Automatic View State Management of ASP.NET Web Forms

```
<%@ Page Description="automatic viewstate management" %>
<%@ Import Namespace = "System.Data" %>

<script language="vb" runat="server">

sub Page_Load(sender as object, args as EventArgs)

    if IsPostBack=false then
        dataGrid.DataSource = CreateDataSource()
        dataGrid.DataBind()
    end if

end sub

sub submit_click(sender as object, args as EventArgs)

    messageDisplay.Text="EnableViewState is set to: " & _
        dataGrid.EnableViewstate.ToString()

end sub

function CreateDataSource() as ICollection

    dim Rand_num as Random = new Random()
    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim dv as DataView
    dim i as Integer

    with dt.Columns
        .Add(new DataColumn("IntegerValue", GetType(Int32)))
        .Add(new DataColumn("StringValue", GetType(string)))
        .Add(new DataColumn("CurrencyValue", GetType(double)))
    end with
```

Listing 7.7 continued

```
for i = 0 to 5 step 1
    dr = dt.NewRow()
    dr(0) = i
    dr(1) = "Item " & i.ToString()
    dr(2) = 1.23 * Rand_Num.Next(1,15)
    dt.Rows.Add(dr)
next

dv = new DataView(dt)

return(dv)

end function

</script>

<html>
<body>
<h2>Automatic ViewState Management</h2>
<hr />

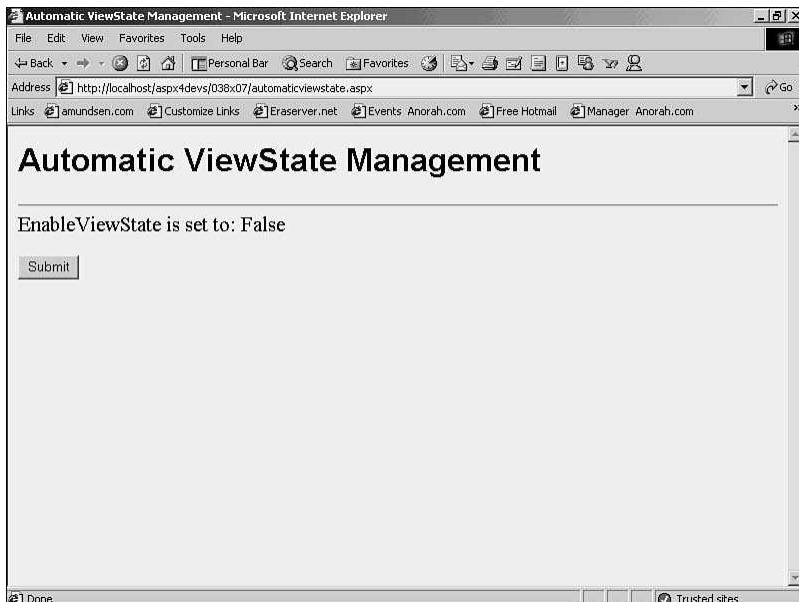
<asp:Label id="messageDisplay" runat="server" />

<form runat="server">
    <asp:Button id="submit" text="Submit"
        OnClick="submit_click" runat="server" />
    <asp:DataGrid id="dataGrid" runat="server" />
</form >

</body >
</html >
```

In Listing 7.7, notice that in the `Page_Load` event, the code that creates the data source and binds it to the grid is executed only the very first time the page is displayed for the user. However, each time the button is clicked on the page, and the page is sent back to the server for processing, it is returned with the grid fully populated. This is possible because ASP.NET Web Forms maintain the view state of the grid in a hidden variable on the page itself.

However, if you add the `EnableViewState="false"` attribute to the `DataGrid` control and restart the page, you'll find that when you press the submit button, the page is sent back without data in the grid (see Figure 7.4).

**Figure 7.4**

Results of setting `EnableViewState="false"`.

TIP

Since enabling and maintaining view state can take some time on large pages with lots of state data (such as data grids and lists), it is possible that turning off view state for some controls can increase performance of your ASP.NET Web Forms applications.

Summary

The ASP.NET Web Forms model helps you build Web pages that are easier to maintain and can be optimized to increase performance in a number of ways:

- You can create your server-side code blocks using the “In-Page” method or the “Code-Behind” method.
- ASP.NET Web Forms allow you to declare client-side events for Web controls and create server-side code blocks to respond to those events.
- You can take advantage of page-level state management to improve the efficiency of your ASP.NET Web Forms.

CHAPTER 8

Creating Simple Web Pages with the HTML Server Controls

In Chapter 7, “Understanding ASP.NET Web Forms,” you learned how the ASP.NET Framework allows you to design Web forms that take advantage of server-side execution and client-side event handlers. You also learned how the Web Form event model works and how you can use it to control the display of your Web pages.

In this chapter, you’ll learn how to use the HTML Server Controls to create simple input and display forms that can participate in the Web Forms events. Basically, you will learn how you can use the common HTML elements used in classic HTML pages (`table`, `input`, `submit`, `select`, and so on) and modify them in a way that will allow you to use server-side VB code to modify their contents and handle their events. In fact, using the techniques you’ll learn in this chapter, you will be able to convert (with only minor modifications) many of your existing, classic HTML forms into ASP.NET Web Forms.

However, before jumping straight into the process of exploring and coding your Web forms using HTML Server Controls, it is a good idea to spend a little time understanding the details of HTML Server Controls themselves and how they work.

What Are HTML Server Controls?

HTML Server Controls are the basic building blocks of simple ASP.NET Web forms. Typically, you would use HTML Server Controls when you want to create Web forms that look and feel very much like classic HTML forms you may have built in the past.

Following is a very simple Web form that allows users to enter a username and password and submit the results to the server. This ASP.NET Web form uses the HTML Server Controls. You should also notice that it looks almost identical to classic HTML forms you already know how to build.

Listing 8.1 Login Form Using HTML Server Controls

```
<form runat="server">
<table>
    <tr>
        <td>Status</td>
        <td id="status" runat="server" />
    </tr>
    <tr>
        <td>Username</td>
        <td><input id="username"
            runat="server" /></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password"
            id="password"
            runat="server" /></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input type="submit" id="submit"
            value="Login" runat="server" />
        </td>
    </tr>
</table>
</form>
```

In Listing 8.1, notice that all the HTML controls on the page are standard HTML elements. No new controls are introduced here. You will also notice that several of the HTML controls have the `runat="server"` attribute. It is this new attribute that turns the common HTML controls into ASP.NET HTML Server Controls.

The Power of the RunAt="Server" Attribute

By simply adding `runat="server"` to standard HTML elements, they become HTML Server Controls. It's just that easy. When you add the `runat="server"` attribute, you are telling the ASP.NET runtime that you want that HTML control to participate in the server-side execution model. In other words, the HTML control is now "known" by the server-side ASP.NET runtime. That means you can use server-side code to read and write attributes of that control and even handle events.

For example, you should notice that there is a `<td>` tag in Listing 8.1 that has an `id` attribute set to "status" and to the `runat="server"` attribute. This means that you can add server-side code to access the `<td>` tag and its attributes. The code snippet below shows how this can be done.

Listing 8.2 Using Server-Side Code to Access the InnerHtml Attribute of a <td> Tag

```
<script runat="server">

Sub Page_Load(Src as Object, E as EventArgs)

    If Not Page.IsPostBack Then
        status.InnerHtml="Not Logged In"
    Else
        status.InnerHtml=<b>Logged In</b>
    End If

End Sub

</script>
```

Listing 8.2 shows the `Page_Load` event of the Web Form. This code first checks to see if the page has been posted back (for example, if the user has pressed the submit button on the page). If the page has not been posted back, the `InnerHtml` attribute of the `<td>` tag is set to "Not Logged In". However, if the page view is due to the user pressing the submit button, the `<td>` tag's `InnerHtml` attribute is set to "Logged In" in bold text. This is an example of how adding the `runat="server"` tag allows you to write server-side code for standard HTML controls.

Why You Can Add RunAt="Server" to Any HTML Tag

You can add the `runat="server"` tag to absolutely any HTML element on the page. For example, you can add it to the `<title>` tag at the top of a Web page:

```
<title id="title" runat="server" />
```

You can then use server-side code to alter the title text of the browser window (see Listing 8.3).

Listing 8.3 Using Server-Side Code to Set the Browser Window Title

```
<script runat="server">

Sub Page_Load(Src as Object, E as EventArgs)

    If Not Page.IsPostBack Then
        title.InnerText="Not Logged In"
    Else
        title.InnerText="Logged In"
    End If

End Sub

</script>
```

The code in Listing 8.3 should look familiar by now. This time, instead of programmatically setting the attributes of a `<td>` tag, Listing 8.3 sets the attribute of the `<title>` tag.

The HTMLControl Class

The reason that you can successfully apply the `runat="server"` attribute to any HTML control is because the ASP.NET Framework contains a special class that is used to define all HTML controls for the server-side runtime. This is called the **HTMLControl Class**. The **HTMLControl Class** is the base class that is used to create all the other **HTMLcontrol classes** in the ASP.NET Framework.

When you add `runat="server"` to any HTML control, the ASP.NET runtime will first attempt to find the correct class to match the HTML control. If it cannot find a class defined for that HTML control, it instead uses the **HTMLControl base class**. In this way, all current and future HTML controls can be supported within the ASP.NET Framework.

Since the **HTMLControl class** is the base class, it supports a limited set of properties and methods. Table 8.1 shows the supported attribute names along with the properties or methods associated with the attribute.

Table 8.1 Properties and Methods of the HTMLControl Base Class

Attribute	Type	Example	Comments
<code>id</code>	Property	<code>id="myspan"</code>	The programmatic name used by the server-side runtime to identify the control on the page.
<code>runat</code>	Property	<code>runat="server"</code>	Marks the control for server-side execution. The only valid value for the <code>runat</code> tag is "server."
<code>disabled</code>	Property	<code>disabled="true"</code>	Adds the <code>disabled</code> attribute to the control when the page is rendered for the browser. Supported only on HTML 4.0-compliant browsers.
<code>style</code>	Property	<code>style="font-family:arial"</code>	Applies one or more Cascading Style Sheet styles to the HTML control.
<code>tagname</code>	Property	<code><input runat="server"/></code>	Returns the tag name of the control. The <code>tagname</code> in the example here is "input".
<code>InnerHtml</code>	Method	<code>Test</code>	Returns (or sets) the HTML contents within the control ("Test").
<code>InnerText</code>	Method	<code>Test</code>	Returns (or sets) only the text within the control ("Test").

It is not important that you memorize these attributes or that you understand completely the relationship between the HTML elements on a Web page and the HTMLControl base class within the ASP.NET Framework. However, it is valuable to know how the ASP.NET Framework uses the base class to provide server-side execution support for your HTML pages.

For the rest of this chapter, you'll learn the complete set of HTML Server Controls in greater detail, including the most commonly used properties, methods, and events available for each of the controls.

The General Controls

The first set of HTML Server Controls to review is a set of general controls. These include controls based on the anchor tag (`<a>`), the image tag (`img`), the form tag (`form`), the division tag (`div`), and the span tag (``). You can use these general controls to build very simple Web pages that are commonly used in almost every Web application.

Anchor (`<A>`)

The Anchor control (`<a>`) is used to add a hyperlink to the document. The code example in Listing 8.4 shows how you can use the anchor tag in an ASP.NET Web form.

NOTE

Throughout the text, you'll see the words "control," "server control," "element," and "tag" used interchangeably. In most cases, they refer to the same item—some markup code within the display portion of the Web page.

Listing 8.4 Using the Anchor HTML Server Control

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

<script runat="server">

Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
        With myAnchor
            .Href="http://www.eraserver.net"
            .Name="myAnchor"
            .Title=".NET Hosting Services"
            .Target="_blank"
            .InnerHtml=<b>eraserver.net</b>"
        End With
    End If

End Sub

</script>
```

Listing 8.4 continued

```
<html>
<body>

<h2>General Controls</h2>
<hr />

<a id="myAnchor" runat="server" />

</body>
</html>
```

Notice that, in the body of the page shown in Listing 8.4, the actual anchor tag (`<a>`) has only two declarative attributes: `id` and `runat`. Notice also the use of the final closing bar at the end of the anchor tag (`/>`). Every HTML control used in ASP.NET pages must have either a closing bar as shown here or a complete closing tag (for example, ``).

You can also see (in Listing 8.4) that the `Page_Load` event is used to populate all the important attributes of the control. Keep in mind that this is server-side code executing here to populate the anchor control.

Image (IMG)

The image control (`img`) is used to display binary images on the page. The following code example (see Listing 8.5) shows how you can use the `img` tag in an ASP.NET Web form.

Listing 8.5 Using the Img HTML Server Control

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

<script runat="server">

Sub Page_Load(Src as Object, _
             E as EventArgs)

    If Not Page.IsPostBack Then

        With myImage
            .Alt="the hand"
            .Align="left"
            .Border=1
            .Height=100
            .Width=79
            .Src="images/hand.gif"
        End With

    End If

```

Listing 8.5 continued

```
End Sub

</script>

<html>
<body>

<h2>General Controls</h2>
<hr />

<img id="myImage" runat="server" />

</body>
</html>
```

Form (Form)

The `form` HTML Server Control works very much like the traditional HTML `form` control. It supports all the common HTML attributes. However, by adding the “`runat="server"`” attribute to the `form` tag, you can greatly increase its power.

In many cases, you will want to create an input form that “posts back” to itself. In other words, after the user enters data and presses the “submit” button, the page refreshes and shows the results of the input in the same page. To do this in classic ASP you would need to add server-side scripting to inspect the `Request` object and add more code in order to copy the values from the `Request` object back into the various input controls within the form.

The `form` HTML Server Control takes advantage of the fact that ASP.NET handles almost all of this work for you. The example below shows how easy it is to create a “postback” form that displays user input results in ASP.NET.

Listing 8.6 An Example of Postback Forms with ASP.NET

```
<form runat="server">

Name: <input id="name" runat="server" />
<input type="submit" runat="server" />

</form>
```

You should notice that the `form` tag in Listing 8.6 contains no `action` or `method` attributes. ASP.NET will automatically supply these for you. If you wish to supply values for the `action` and `method` attributes, you can still do so.

You should also notice that there is absolutely no server-side code in this example. However, when you run the example, you’ll find that any value entered into the input control by the user will be displayed in that same input control after the page has posted. This persisting of input values is another automatic aspect of ASP.NET.

Division (Div) and Span (Span)

You've already seen several examples of using the span Server Control in ASP.NET pages. The div and span controls allow you to "name" spaces within the Web form and then use server-side code to inspect or update the HTML or text contents of those named spaces.

The div and span controls support the following additional attributes:

- **InnerHtml** allows you to insert any standard HTML into the body of the tag.
- **InnerText** allows you to insert plain text into the body of the tag without disturbing any existing HTML encoding (bold, italic, and so on) that is already contained in the body of the tag.

Following is an example that inserts the results of a user input in a named div tag.

Listing 8.7 Using Div Tags in ASP.NET to Update Page Contents

```
<script runat="server">

Sub Page_Load(sender As Object, _
E As EventArgs)

If Page.IsPostBack=True Then
    msgDiv.InnerHtml = "<b>" & _
        name.value & "</b>"
End If

End Sub

</script>

<body>

<h3>Post Back Form</h3>

<div id="msgDiv" runat="server" />

<form runat="server">

Name: <input id="name" runat="server" />
<input type="submit" runat="server" />

</form>

</body>
```

When you run the preceding code in your browser, enter a value into the input box and press the submit button. You'll see results that match those in Figure 8.1.

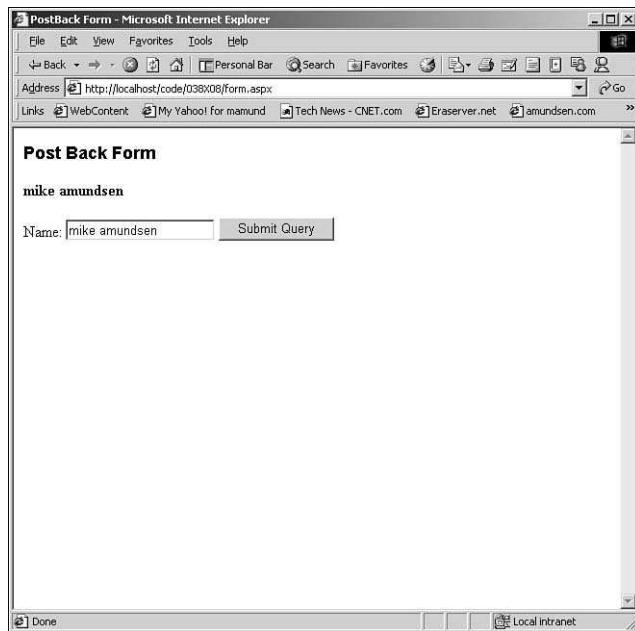


Figure 8.1

Results of a postback form in ASP.NET.

The Table Controls

The HTML Server Controls also support the full range of HTML table elements including headers (**th**), rows (**tr**), and detail cells (**td**). Since ASP.NET allows you to mark any of these controls with the **runat="server"** attribute, you can also use server-side code to easily manipulate tables within an ASP.NET page.

Table

Listing 8.8 Using Server-Side Code to Set Table Attributes

```
<script runat="server">

Sub Page_Load(sender As Object, _
E As EventArgs)

    With table
        .bgColor = "lightblue"
        .border = "1"
    End With

End Sub

</script>
```

Listing 8.8 continued

```
<body>

<h3>Tables</h3>

<table id="table" runat="server">
    <tr>
        <td>Hello!</td>
    </tr>
</table>

</body>
```

Table Header (Th), Row (Tr), and Detail (Td)

You can also use server-side code to manipulate the th, tr, and td tags of a table. The following code shows how this can be done.

Listing 8.9 Manipulating Th, Tr, and Td Tags Using Server-Side Code

```
<script runat="server">

Sub Page_Load(sender As Object, _
    E As EventArgs)

    With table
        .bgColor = "lightblue"
        .border = "1"
    End With

    th.InnerText="Title"
    tr.bgcolor="lightyellow"
    td.InnerHtml=<i>New Content</i>"

End Sub

</script>

<body>

<h3>Tables</h3>

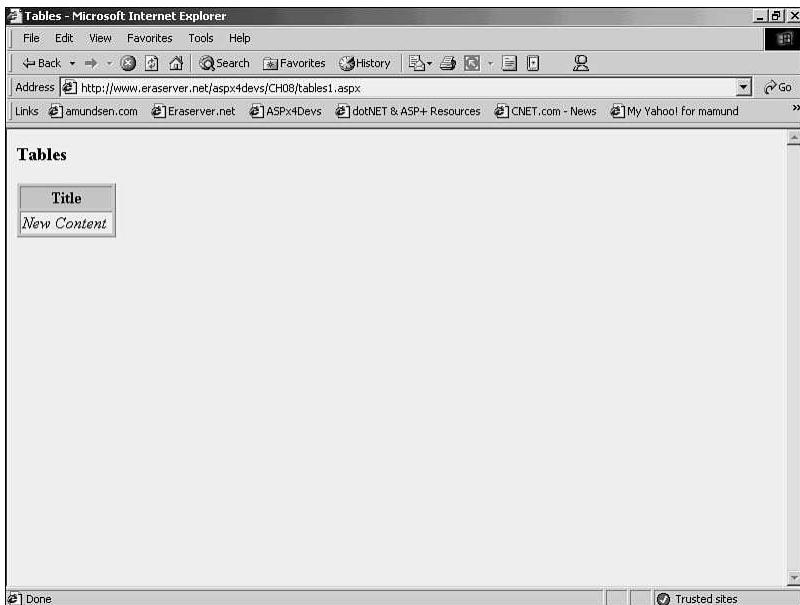
<table id="table" runat="server">
    <thead>
        <tr>
            <th id="th" />
        </tr>
    </thead>
```

Listing 8.9 continued

```
</thead>
<tbody>
    <tr id="tr" runat="server">
        <td id="td"
            runat="server">Hello!</td>
    </tr>
</tbody>
</table>

</body>
```

When you run the code in Listing 8.9 in your browser, it should look like the example in Figure 8.2.

**Figure 8.2**

Results of applying server-side code to table tags.

The Input Controls

As you would assume, the HTML Server Controls support the complete range of HTML input controls including text input, buttons, and lists. In most cases, you can use the same HTML coding that you would use in standard HTML pages. However, you can also use a number of handy server-side code methods and properties to easily populate and inspect the contents of Server Control input elements.

ASP.NET also has very good support for the HTML 4.0 file input control. With ASP.NET, you no longer need to install a separate upload component to support file uploads in your Web applications.

Text, Password, Textarea, and Hidden

Just as in standard HTML pages, ASP.NET's Server Controls provide a full range of input controls to work with. This includes the `text`, `password`, and `textarea` type controls as well as the `hidden` type. By combining typical HTML elements and ASP.NET server-side code, you can build a sign-in form like the one in Listing 8.10.

Listing 8.10 Using Various Input Controls with ASP.NET

```
<body>

<h2>User Login</h2>
<hr />

<form runat="server">
<input type="hidden" id="hidden" runat="server" />
<table>
    <tr>
        <td>Status</td>
        <td id="status" runat="server" />
    </tr>
    <tr>
        <td>Username</td>
        <td><input id="username"
               runat="server" /></td>
    </tr>
    <tr>
        <td>Password</td>
        <td><input type="password"
               id="password"
               runat="server" /></td>
    </tr>
    <tr>
        <td>&nbsp;</td>
        <td><input type="submit" id="submit"
               value="Login" runat="server" />
        </td>
    </tr>
</table>

<textarea id="textarea" runat="server" />
```

Listing 8.10 continued

```
</form>
```

```
</body>
```

After creating the ASP.NET form, you can then add a server-side code block to handle the initial form presentation and response to the user login. In this example, no checks are made for a valid username or password. This example just shows you how to code against the various input controls.

Listing 8.11 Server-Side Code Block to Support the Login Form

```
<script runat="server">

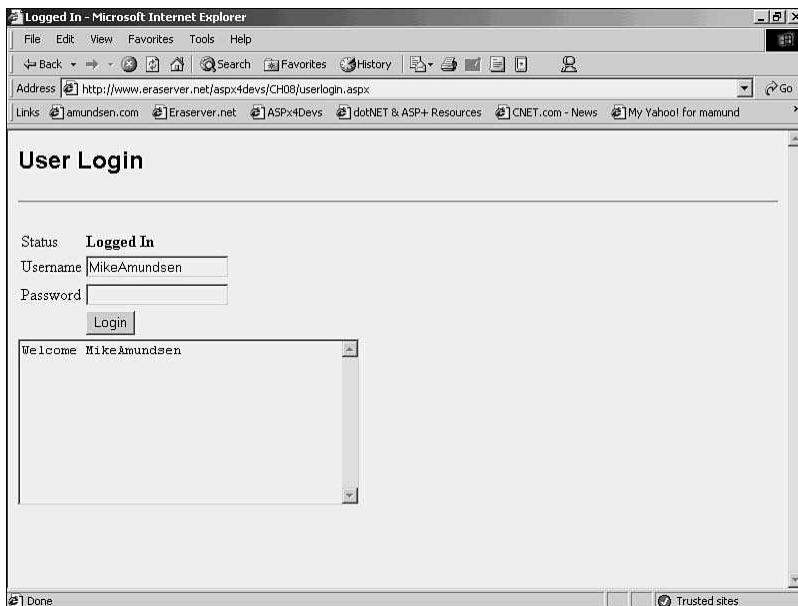
Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
        ' output before log-in
        status.InnerHtml= _
            "<b>Not Logged In</b>"
        hidden.value = "unknown"
        textarea.visible = false
    Else
        ' output after log-in
        status.InnerHtml=<b>Logged In</b>
        hidden.value = username.value

        with textarea
            .InnerText = "Welcome " & _
                hidden.value
            .visible = true
            .cols=40
            .rows=10
        end with

    End If
End Sub

</script>
```

When you first load this form in your browser, you will only see the header, two prompts and input boxes, and the login button. After filling in the input controls and pressing the submit button, you'll see the `textarea` control appear with text. Figure 8.3 shows how the form appears after successfully logging in.

**Figure 8.3**

Results of logging into the form.

Submit, Reset, Image, and Button

The Server Controls also provide several ways to capture a button-click event. Typically this is done within an HTML `form` element using a submit button. However, you can also use an image type input control to submit a form. There is also support for a reset type input and a generic button element.

This is a good time to discuss the differences in the way ASP.NET forms treat the various buttons. First, as you saw earlier in Listing 8.10, you can add a fully functional “submit” type button to a server form without needing to create any special server-side code to listen for the submit event. The details of “wiring up” the event handler for the submit button are handled automatically by the ASP.NET Web Forms environment.

This is also true for the image-type input button. You can simply add this input control to the form and it will work just like the submit-type button. However, if you want to perform a reset of the form, you will want to go a bit farther.

Listing 8.12 shows an example using both the submit- and image-type buttons in a form.

Listing 8.12 Using a Submit and Image Button in an ASP.NET Web Form

```
<form runat="server">

<input type="text" id="textInput" runat="server" />

<p>
<input type="submit"
   id="submit"
   runat="server"
/>
</p>

<p>
<input type="image"
   src="images/poweredbymyself.gif"
   id="image" runat="server"
/>
</p>

</form>
```

In Listing 8.12, clicking on either the image or the submit button will post the form back to the server.

Reset and HTML 4.0 Buttons

In standard HTML forms, the reset button can be used to restore all the inputs on the page to their original settings. In effect, this allows users who have made a series of inputs on the form to change their minds and—in one click—reset the form to its starting values. In ASP.NET, the “reset”-type button does not perform quite as you would expect. This is due to ASP.NET’s automatic state management feature that automatically remembers and restores the values of inputs on postback. To make sure it works in all situations, you should register a server-side event handler for the reset button and explicitly clear all the inputs.

Listing 8.13 shows how this can be done in the HTML portion of the page.

Listing 8.13 Registering a Server-Side Event Handler for an HTML Server Control Button

```
<p>
<input type="reset"
   OnServerClick="reset_Click"
   runat="server"
/>
</p>
```

Listing 8.13 continued

```
<p>
<button
    OnServerClick="reset_click"
    runat="server">
Clear Input
</button>
</p>
```

You should note that the new attribute added to the HTML element is `OnServerClick`. With the HTML Server Controls, you can register both a client-side and server-side event handler. This allows you the flexibility to perform client-side processing, server-side processing, or both.

Listing 8.14 shows the code block to respond to the server-side event.

Listing 8.14 Server-Side Code to Respond to a Button Click Event

```
<script runat="server">

Sub reset_click(Src as Object, _
    E as EventArgs)
    textInput.value = ""
    postMsg.InnerText = "Reset at " & now()
End Sub

</script>
```

Select

The HTML Server Controls also support the HTML select control to create drop-down boxes and list controls. You can use server-side code to populate the controls and also show contents of the list control after the user submits the form.

Listing 8.15 shows an example of a simple drop-down list, populated using server-side code, along with a server-side event handler to display the selected list item in the form.

Listing 8.15 Manipulating the Select Control Using ASP.NET Server Controls

```
<script runat="server">

Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
```

Listing 8.15 continued

```
With sportsList
    .Items.Add("Football")
    .Items.Add("Soccer")
    .Items.Add("Baseball")
    .selectedIndex=2
End With

Else

End If

End Sub

Sub submit_click(Src as Object, E as EventArgs)

    postMsg.InnerText = "Sport selected: " _
        & sportsList.value

End Sub

</script>

<body>

<h2>Lists, RadioButtons, CheckBoxes</h2>
<hr />

<form runat="server">

<div id="postMsg" runat="server" />

<select id="sportsList" runat="server"/>

<input type="submit" OnServerClick="submit_click"
runat="server" />

</form>

</body>
```

When this form runs, you'll see that three items have been added to the list, and the last item in the list is the current selection. When the user clicks the submit button, the form will display the user's selection.

Radio and Checkbox

You can also use standard radio buttons and checkboxes in your ASP.NET Web forms. Listing 8.16 shows a form with three radio buttons and one checkbox element. You can also see the server-side code that is used to inspect the posted results when the user clicks on the submit button.

Listing 8.16 Using Radio Buttons and Checkboxes in HTML Server Control Web Forms

```
<script runat="server">

Sub Page_Load(Src as Object, E as EventArgs)

    Dim replyMsg as String = ""

    If Not Page.IsPostBack Then

        living.checked=false
        cat.checked=true

    Else

        If Cat.checked=true Then
            replyMsg="cat"
        End If

        If Dog.checked=true Then
            replyMsg="dog"
        End If

        If Bird.checked=true Then
            replyMsg="bird"
        End If

        If living.checked=true Then
            postMsg.InnerText="I hope you and your " & _
                replyMsg & " are happy!"
        Else
            postMsg.InnerText="I am sorry you to not have a favorite pet " & _
                replyMsg & " right now"
        End IF

    End If

End Sub

</script>
```

Listing 8.16 continued

```
<body>

<h2>Radio and Checkbox Inputs</h2>
<hr />

<form runat="server">

<div id="postMsg" runat="server" />

<input type="radio" id="cat"
       name="pet" runat="server" /> Cat
<input type="radio" id="dog"
       name="pet" runat="server" /> Dog
<input type="radio" id="bird"
       name="pet" runat="server" /> Bird

<p>
Are you currently living with your
selected pet?
<input type="checkbox"
       id="living" runat="server" />
</p>

<p>
<input type="submit" runat="server" />

</form>

</body>
```

Notice how the three radio-type controls all have the same name attribute setting (pet). This marks the radio buttons as a group and makes sure that only one of the three radio buttons can be selected in the form.

File Input

The last HTML Server Control to cover in this chapter is the File Input control. This is an HTML 4.0 control that allows users to select a file on their workstation and mark it for upload to the server. In previous versions of ASP, you needed to install and enable a custom component that would accept the incoming file and store it in the proper directory on the server.

ASP.NET, however, ships with all the components needed to handle file uploads to your server. All you need to do is add some server-side code to handle the file posting, save it to the proper folder, and optionally notify the user that the file has been saved.

Listing 8.17 shows how simple this is with ASP.NET.

Listing 8.17 Completing a File Upload with ASP.NET

```
<script runat="server">

Sub Submit_Click(Src as Object, _
E as EventArgs)

    uploadFile.PostedFile.SaveAs("c:\\" & _
        saveFile.Value)
    postMsg.InnerHtml = "File uploaded to " & _
        "<b>c:\\" & saveFile.Value & _
        "</b> on the web server"

End Sub

</script>

<body>

<h2>File Upload</h2>
<hr />

<form enctype="multipart/form-data" runat="server">

    <p>
        <span id=postMsg runat="server" />
    </p>

    <p>
        Select File to Upload:
        <input id="uploadFile" type=file
            runat="server" />
    </p>

    <p>
        Save as filename (no path):
        <input id="saveFile" type="text"
            runat="server" />
    </p>

    <p>
        <input type=button id="submit"
            value="Upload"
            OnServerClick="Submit_Click" runat="server" />
    </p>

</form>
```

When you load this Web page on your browser, you can select a file from your workstation and upload it to the server where it will be stored in the root folder of the C: drive (see Figure 8.4).

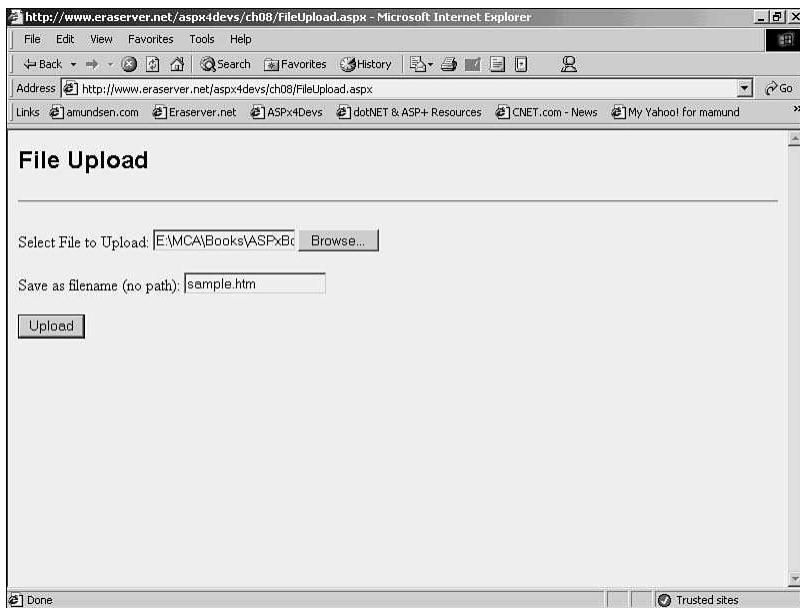


Figure 8.4

Uploading a file to the server.

Summary

ASP.NET uses the HTML Server Control library to easily create Web forms that use the standard HTML controls with which most Web programmers are familiar. You can simply add the `runat="server"` attribute to any HTML control in order to use ASP.NET server-side code to manipulate the control attributes and access ASP.NET's events and methods.

To recap, here are the important controls covered in this chapter:

- The anchor (`<a>`), image (`img`), form (`form`), division (`div`), and span (`span`) controls are used to create simple form layouts.
- The table controls (`table`, `th`, `tr`, and `td`) are used to enhance the layout of forms.
- The input controls (`text`, `password`, `textarea`, `select`, `radio`, and `checkbox`) are used to build forms that interact with the user.
- The File Input control is used to create a form that takes advantage of the built-in file upload services found in the .NET Framework.

In the next chapter, you'll learn how to use the new ASP.NET "smart controls" to create even more powerful and flexible Web pages.

CHAPTER 9

Creating Interactive Forms with Web Form Server Controls

In Chapter 8, “Creating Simple Web Pages with the HTML Server Controls,” you took a look at creating pages employing HTML server controls. The HTML server controls provide a set of controls that are close analogs of the standard HTML elements. Microsoft provides an alternative set of server controls, the Web server controls, that you can also use to create ASP.NET Web pages. Unlike the HTML server controls, the Web server controls have a rich, strongly typed object model and include support for advanced features such as data binding, templates, and automatic browser detection.

There are three basic groups of Web server controls: standard controls, rich controls, and validation controls. In this chapter, you’ll take a look at the standard Web server controls in detail. You’ll learn how to employ the Web server controls in your pages and take advantage of their object models.

What Are Web Server Controls?

As you learned in Chapter 8, the HTML server controls have properties and methods that mirror the attributes of the HTML elements. This makes it very easy to migrate classic ASP pages employing standard HTML form controls and elements to ASP.NET pages employing the HTML server controls. In many cases, it’s as simple as adding a runat attribute and setting it equal to “server.”

In contrast to the HTML server controls, the Web server controls have an abstracted object model that abandons any

attempt to closely map the HTML elements. Control attributes differ from the HTML element attributes. In addition, ASP.NET controls don't map one-to-one with the HTML elements. For example, the HTML server controls have a single control for both list and dropdown controls: `HTMLSelect`, which maps to the `SELECT` element. The Web server controls, however, provide separate controls: `ListBox` and `DropDownList`.

As already mentioned, there are three basic categories of Web server controls: standard controls, rich controls, and validation controls. The standard controls can be classified further into three subgroups: the general controls, the form controls, and the table controls.

You use the `asp:` tag prefix to refer to Web server controls. The `asp:` tag prefix, along with the `runat="server"` attribute, tells ASP.NET that you wish to use a Web server control rather than a standard HTML control. For example, to add a `Label` Web server control to the page, you would use the following HTML:

```
<asp:label id="lblCity" text="City" runat="server">
```

The WebControl Class

With a few exceptions, the Web server controls derive from the `System.Web.UI.WebControls.WebControl` class. Table 9.1 lists the most common properties that all Web controls have in common. Most controls have properties in addition to these common properties.

Table 9.1 Common Properties of Web Server Controls

Property	Description
<code>AccessKey</code>	Keyboard shortcut key for the control
<code>Attributes</code>	Collection of arbitrary attributes of the control
<code>BackColor</code>	Background color of the Web control
<code>BorderColor</code>	Border color of the control
<code>BorderStyle</code>	Border style of the control
<code>BorderWidth</code>	Border width of the control
<code>CssClass</code>	CSS class of the control
<code>Enabled</code>	If true, the control is enabled; if false, the control is disabled
<code>Font</code>	Retrieves font information for the control
<code>ForeColor</code>	Foreground color of control text
<code>Height</code>	Height of the control
<code>Style</code>	CSS style attribute of the control
<code>TabIndex</code>	Tab index of the control
<code>ToolTip</code>	Tool tip for the control to be displayed when user hovers over the control
<code>Width</code>	Width of the control

General Controls

The general Web server controls include several general-purpose controls: the HyperLink, LinkButton, Image, Label, and Panel controls.

HyperLink

You use the HyperLink Web server control to create a hyperlink. The HyperLink control maps to the HTML `<a>` (or anchor) element. The page shown in Listing 9.1 illustrates the use of the Hyperlink control.

Listing 9.1 WEBHYPERLINK.ASPX—Using the `HyperLink` Server Control

```
<%@ Page Language="vb" %>
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    hlk.NavigateURL = "http://www.aspxdev.com"
    hlk.ToolTip = "You read the book, now visit the site!"
    hlk.Text = "ASP.NET for Developers Web Site"
End Sub
</script>
</head>

<body>
<form runat="server">
    <asp:HyperLink id="hlk" runat="server" />
</form>
</body>
</html>
```

As you can see in Listing 9.1, the `NavigateURL` property of the `Hyperlink` control maps to the `<a>` element's `HREF` attribute; you use it to specify the URL to which you wish to point the hyperlink.

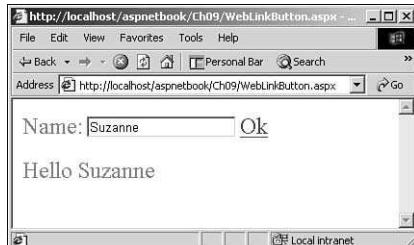
LinkButton

The `LinkButton` control has the appearance of a hyperlink but the behavior of a button control. When clicked, the `LinkButton` control causes a form post back to occur. You can also attach code to the `OnClick` event of the control as shown in Listing 9.2. The page in Listing 9.2 can be seen in Figure 9.1.

Listing 9.2 WEBLINKBUTTON.ASPX—Caption

```
<%@ Page Language="vb" %>
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    a {font-size: 18pt;}
</style>
<script language="VB" runat="server">
Sub LinkClick(Src as Object, E as EventArgs)
    lblOut.Text = "Hello " & txtName.Text
End Sub
Sub Page_Load(Src as Object, E as EventArgs)
    lbtOk.Text = "Ok"
End Sub
</script>
</head>

<body>
<form runat="server">
    <p><asp:Label id="lblName" runat="server" text="Name:" />
        <asp:TextBox id="txtName" runat="server" />
    <asp:LinkButton id="lbtOk" runat="server" onclick="LinkClick" /></p>
    <p><asp:Label id="lblOut" runat="server" /></p>
</form>
</body>
</html>
```

**Figure 9.1**

Using the LinkButton Server control.

ASP.NET emits client-side JavaScript code to make the LinkButton control work. Thus, the LinkButton control will not work when used with a browser that doesn't support JavaScript.

Image

The `Image` server control maps to the HTML `` element. You use it to display an image on the page. You set the source of the image using the `ImageUrl` property. The code shown in Listing 9.3 illustrates the use of the `Image` control.

Listing 9.3 WebImage.aspx—You Can Use the Image Server Control to Display an Image on the Page

```
<%@ Page Language="vb"%
<html>
<head>
<style>
    span {font-size: 18pt; color: blue;}
    a {font-size: 18pt;}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    With imgAspConnections
        .ImageUrl = "aspconnectionslogo.gif"
        .AlternateText = "Microsoft ASP.NET Connections Conference"
        .ImageAlign = ImageAlign.Right
    End With
End Sub
</script>
</head>

<body>
<form runat="server">
    <asp:Image id="imgAspConnections" runat="server" />
    <asp:Label id="lblText" runat="server"
        text= "Microsoft ASP.NET Connections Conference
                is a conference for ASP.NET developer."/>
</form>
</body>
</html>
```

As shown in this example, you can specify alternate text (mapped to the `` element's `ALT` attribute) using the `AlternateText` property. You can also use the `ImageAlign` property (mapped to the `` element's `ALIGN` attribute) to align the image to elements on the page that follow it. When set from code, you must set the `ImageAlign` property to one of the `ImageAlign` enumeration values (for example, `ImageAlign.Left`, `ImageAlign.Right`, and so on).

NOTE

The `Image` server control does not respond to `click` events. Use the `ImageButton` control if you need to respond to `click` events.

Label

There's not much to it, but the Label server control is the workhorse of the Web server controls. You can use it to display static information and informational messages in response to the events of other controls. You use the Text property of the Label control to set its text. Label controls are mapped to HTML `` tags.

Most of the examples in this chapter employ one or more label controls, including the code shown in Listing 9.3.

Panel

You can use the Panel server control to serve as a container for other controls. You might use the Panel control to group a number of controls on top of a background image or swatch of color, or to easily hide or show a set of controls.

The Panel control has a number of properties that you can use to control various attributes such as the style, color, and width of the border, and background color. The code in Listing 9.4 demonstrates the use of some of these properties in creating two Panel controls.

Listing 9.4 WebPanel.aspx—The Panel Server Control Is Useful for Hiding and Showing a Group of Controls

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    If Page.IsPostBack Then
        pnlControls.Visible = Not (chkHide.Checked)
        pnlControls.Enabled = Not (chkDisable.Checked)
    End If
End Sub
</script>
</head>

<body>
<form runat="server">

    <asp:Panel id="pnlChecks" runat="server"
        backcolor="#99ccff" borderstyle="solid"
        borderwidth="1" bordercolor="#0000ff"
        Height="40px" Width="220px">
        <asp:Checkbox id="chkHide" runat="server"
            autopostback="True" text="Hide Controls" /> <br />
        <asp:Checkbox id="chkDisable" runat="server"
            autopostback="True" text="Disable Controls" />
    </asp:Panel>
```

Listing 9.4 continued

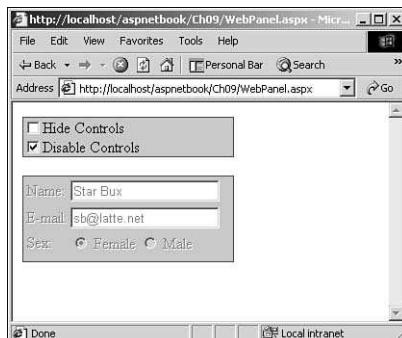
```

<p>
<asp:Panel id="pnlControls" runat="server"
    backcolor="#99ccff" borderstyle="solid"
    borderwidth="1" bordercolor="#0000ff"
    height="90px" width="220px">
    <table>
        <tr>
            <td>Name:</td>
            <td><asp:Textbox id="txtName" runat="server" /></td>
        </tr>
        <tr>
            <td>E-mail:</td>
            <td><asp:Textbox id="txtEmail" runat="server" /></td>
        </tr>
        <tr>
            <td>Sex:</td>
            <td><asp:RadioButton id="radFemale"
                groupname="sex" runat="server" /> Female
                <asp:RadioButton id="radMale"
                groupname="sex" runat="server" /> Male</td>
        </tr>
        </table>
    </asp:Panel>

</form>
</body>
</html>

```

The first Panel control, pnlChecks, on the WebPanel.aspx page contains two Checkbox controls which are used to change the Visible and Enabled properties of the second Panel control, pnlControls. As you can see from Listing 9.4, Panel controls can contain just about any other control, including both Web and HTML controls. The Webpanel.aspx page is shown in Figure 9.2.

**Figure 9.2**

This page contains two Panel controls.

Depending upon the browser, Panel server controls map to either <div> or <table> HTML controls.

Form Controls

ASP.NET has a number of Web server form controls you can employ to create interactive Web forms. The Web server form controls include the TextBox, RadioButton, CheckBox, DropDownList, ListBox, Button, and ImageButton controls.

In order for your event handling code to work properly, you must enclose Web server controls within a <form> element that includes the runat="server" attribute.

NOTE

In cases where you wish a form to "post away" to another form, you won't want to include the runat="server" attribute in the <form> element. This is discussed in more detail in the section "Creating Post Away Forms" later in this chapter.

TextBox

The TextBox Web server control is a flexible control that you can configure to support single-line, multi-line or password modes. The unique properties of the TextBox control are summarized in Table 9.2.

Table 9.2 Selected TextBox Control Properties

Property	Description
AutoPostBack	Controls whether a change to the contents of a textbox are automatically posted back to the server. To implement this feature, the browser must support JavaScript. True or False (default).
Columns	The width of the TextBox, in characters.
MaxLength	The maximum number of characters that may be entered into the control. Does not apply for multiline TextBox controls.
ReadOnly	Controls whether the contents of the TextBox control are readonly. True or False (default).
Rows	The height of the TextBox, in characters. Only applicable when TextMode is set to TextBoxMode.MultiLine.
Text	The contents of the TextBox control.
TextMode	Controls how the TextBox control behaves. When set to SingleLine, the control behaves like a standard text-style <input> control. When set to MultiLine, the control behaves like a <textarea> control. When set to Password, the control behaves like a password-style <input> control.
Wrap	Controls whether words are wrapped to the next line of a multiline control. True (default) or False.

TIP

When you set the value of `TextBoxMode` from an HTML attribute, you set it to one of the following strings: `SingleLine`, `MultiLine`, or `Password`. When you set the value of `TextBoxMode` from code, however, you must set it to one of the `TextBoxMode` enumeration values: `TextBoxMode.SingleLine`, `TextBoxMode.MultiLine`, or `TextBoxMode.Password`.

The code in Listing 9.5, from `WebTextBox.aspx`, illustrates the use of several of the `TextBox` control properties to produce four different `TextBox` controls.

Listing 9.5 WebTextBox.aspx—The TextBox Server Control Can Be Used in a Number of Different Ways

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: yellow}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
        txtSL2.TextMode = TextBoxMode.SingleLine
        txtSL2.ReadOnly = True
        txtSL2.Text = "Can't be modified"
    End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <table>
        <tr>
            <td>SingleLine TextBox:</td>
            <td><asp:Textbox id="txtSL1" runat="server"
                TextMode="Singleline" Columns=20 /></td>
        </tr>
        <tr>
            <td>SingleLine ReadOnly TextBox:</td>
            <td><asp:Textbox id="txtSL2" runat="server" /></td>
        </tr>
        <tr>
            <td>MultiLine TextBox:</td>
            <td><asp:Textbox id="txtML1" runat="server"
                TextMode="Multiline" Columns=20 Rows=5 /></td>
        </tr>
        <tr>
```

Listing 9.5 continued

```
<td>Password TextBox:</td>
<td><asp:Textbox id="txtPW1" runat="server"
    TextMode="Password" Columns=20 /></td>
</tr>
</table>
</form>
</body>
</html>
```

The page produced by WebTextBox.aspx is shown in Figure 9.3.

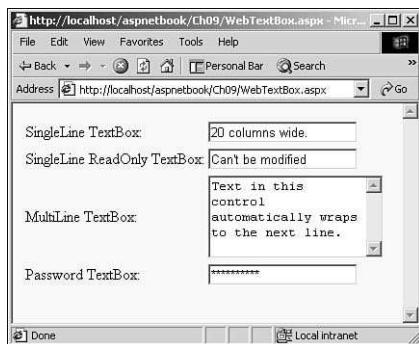


Figure 9.3

Variations of TextBox server controls.

The TextBox server controls support one event: `TextChanged`. `TextChanged` is fired anytime you change the contents of the control and move away from the control. If the `AutoPostBack` property is set to `True`, then `TextChanged` fires immediately. Otherwise, the event happens when the form is posted. The code in Listing 9.6 illustrates the use of the `TextChanged` event with `AutoPostBack` enabled.

Listing 9.6 WebTextBox2.aspx—Using the AutoPostBack Property of a TextBox Control

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub SayHello(Src as Object, E as EventArgs)
    lblMsg.Text = "Hello " & txtName.Text
End Sub
</script>
```

Listing 9.6 continued

```
</head>

<body>
<form runat="server">
    Name:
    <asp:Textbox id="txtName" runat="server"
        TextMode="Singleline" Columns=20
        AutoPostBack=True OnTextChanged="SayHello" />
    <br />
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

TIP

Unless you are using the AutoPostBack feature, there's not much advantage to placing your event handling code in a `TextChanged` event handler instead of the `Page_Load` event handler.

RadioButton

You can use the `RadioButton` Web server control to add radio buttons to a Web page. While you can place a single `RadioButton` control on a page, more typically you will want to group several mutually-exclusive `RadioButton` server controls together on a page to offer a list of choices to the user. You can group a set of `RadioButton` controls together and make them mutually exclusive by setting the `GroupName` property of the `RadioButton` controls to the same name.

You can use the `RadioButton` control's `Text` property to supply labels for the generated radio button.

In order to determine which `RadioButton` control in a group of controls has been checked, you must check each `RadioButton` control's `Checked` property. For example, the code in Listing 9.7 illustrates how to determine which `RadioButton` control in a group of controls was selected.

Listing 9.7 WebRadioBtn.aspx—Mutually Exclusive RadioButton Controls

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: yellow}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
```

Listing 9.7 continued

```
If Not Page.IsPostBack Then
    txtSL2.TextMode = TextBoxMode.SingleLine
    txtSL2.ReadOnly = True
    txtSL2.Text = "Can't be modified"
End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <table>
        <tr>
            <td>SingleLine TextBox:</td>
            <td><asp:Textbox id="txtSL1" runat="server"
                TextMode="Singleline" Columns=20 /></td>
        </tr>
        <tr>
            <td>SingleLine ReadOnly TextBox:</td>
            <td><asp:Textbox id="txtSL2" runat="server" /></td>
        </tr>
        <tr>
            <td>MultiLine TextBox:</td>
            <td><asp:Textbox id="txtML1" runat="server"
                TextMode="Multiline" Columns=20 Rows=5 /></td>
        </tr>
        <tr>
            <td>Password TextBox:</td>
            <td><asp:Textbox id="txtPW1" runat="server"
                TextMode="Password" Columns=20 /></td>
        </tr>
    </table>
</form>
</body>
</html>
```

TIP

As an alternative to checking the values of each **RadioButton** control in a group, you can determine the value of the selected **RadioButton** control in a group of **RadioButton** controls by reading the value of `Request.Form ("group_name")`.

RadioButton controls have an event, `CheckedChanged`, that works similarly to the **TextBox** control's `TextChanged` event. And, like the **TextBox** control, the **RadioButton** control supports the `AutoPostBack` property. The code in Listing 9.8 is functionally equivalent to the code from Listing 9.7, but uses the `CheckedChanged` event along with the `AutoPostBack` property instead of the `Page_Load` event.

Listing 9.8 WebRadioBtn2.aspx—RadioButton Controls Trigger the CheckedChanged Event

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub GetShip(Src as Object, E as EventArgs)
    lblMsg.Text = "You have selected "
    If radUPS.Checked Then
        lblMsg.Text &= "United Parcel Service"
    Else If radFedEx.Checked Then
        lblMsg.Text &= "Federal Express"
    Else If radAirborne.Checked Then
        lblMsg.Text &= "Airborne Express"
    Else If radUSPS.Checked Then
        lblMsg.Text &= "United States Postal Service"
    End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <b>Shipping Method</b><br />
    <asp:RadioButton id="radUPS" runat="server"
        GroupName="grpShip" Text="UPS"
        OnCheckedChanged="GetShip" AutoPostBack=True/>
    <br />
    <asp:RadioButton id="radFedEx" runat="server"
        GroupName="grpShip" Text="FedEx"
        OnCheckedChanged="GetShip" AutoPostBack=True/>
    <br />
    <asp:RadioButton id="radAirborne" runat="server"
        GroupName="grpShip" Text="Airborne"
        OnCheckedChanged="GetShip" AutoPostBack=True/>
    <br />
    <asp:RadioButton id="radUSPS" runat="server"
        GroupName="grpShip" Text="USPS"
        OnCheckedChanged="GetShip" AutoPostBack=True/>
    <br>
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

Notice that the RadioButton controls in Listing 9.8 have their AutoPostBack property set to True. AutoPostBack requires the browser to support JavaScript.

NOTE

ASP.NET supports another Web server control, the RadioButtonList control which lets you bind a list of radio buttons to a data source. The RadioButtonList control is discussed in Chapter 10, “Designing Advanced Interfaces with Web Form List Controls.”

CheckBox

You can use CheckBox Web server controls to add check box controls to a page. You can use the CheckBox control’s Text property to supply a label for the control. The CheckBox control’s Checked property is set to True when the check box is checked and False when it is unchecked.

Listing 9.9 illustrates the use of five CheckBox controls to collect information on five true/false attributes.

Listing 9.9 WebCheckBox.aspx—You Need to Check the Checked Property of a CheckBox Control to Determine If It Has Been Checked

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    If Page.IsPostBack Then
        lblMsg.Text = ""
        If chkRich.Checked Then
            lblMsg.Text &= "rich & "
        End If
        If chkFamous.Checked Then
            lblMsg.Text &= "famous & "
        End If
        If chkFunny.Checked Then
            lblMsg.Text &= "funny & "
        End If
        If chkSmart.Checked Then
            lblMsg.Text &= "smart & "
        End If
        If chkHappy.Checked Then
            lblMsg.Text &= "happy & "
        End If
    If lblMsg.Text.Length = 0 Then
```

Listing 9.9 continued

```

lblMsg.Text = "You are none of the above!"
Else
    lblMsg.Text = "You are " & _
        lblMsg.Text.Substring(0, lblMsg.Text.Length-3) & _
        " !"
End If
End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <asp:CheckBox id="chkRich" runat="server"
        Text="Rich" />
    <br />
    <asp:CheckBox id="chkFamous" runat="server"
        Text="Famous" />
    <br />
    <asp:CheckBox id="chkFunny" runat="server"
        Text="Funny" />
    <br />
    <asp:CheckBox id="chkSmart" runat="server"
        Text="Smart" />
    <br />
    <asp:CheckBox id="chkHappy" runat="server"
        Text="Happy" />
    <br />
    <asp:Button id="cmdSubmit" runat="server"
        text="Done!" />
    <br>
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>

```

The WebCheckBox.aspx page is shown in Figure 9.4.

Like the RadioButton control, the CheckBox server control supports the AutoPostBack property and the CheckedChanged event.

DropDownList

The DropDownList Web server control provides a great way for users to choose items from a list in a small amount of space. You can populate the list of items using ListItem server controls like this:



Figure 9.4

CheckBox server controls are great for asking true/false type questions.

```
<asp:DropDownList runat="server" >
  <asp:ListItem value="value 1">item 1</asp:ListItem>
  <asp:ListItem value="value 2">item 2</asp:ListItem>
  <asp:ListItem value="value 3">item 3</asp:ListItem>
</asp:DropDownList>
```

You can also populate a DropDownList server control by adding `ListItem` objects to the `DropDownList` control's `Items` collection using the `Items.Add` method in code like so:

```
Sub Page_Load(Src as Object, E as EventArgs)
  If Not Page.IsPostBack Then
    DropDownList.Items.Add(New ListItem("item 1", "value 1"))
    DropDownList.Items.Add(New ListItem("item 2", "value 2"))
    DropDownList.Items.Add(New ListItem("item 3", "value 3"))
  End If
End Sub
```

NOTE

You can also populate `DropDownList` and `ListBox` controls using data binding, which is discussed in Chapter 10, “Designing Advanced Interfaces with Web Form List Controls.”

You can retrieve the selected `ListItem` from a `DropDownList` control using the `DropDownList.SelectedItem` property, which returns a `ListItem` object. From there, you can retrieve the text and value associated with the particular `ListItem` using the `Text` and `Value` properties, respectively.

The `DropDownList` control has an event, the `SelectedIndexChanged` event, which is fired when the user selects an item from the list. Like the other form controls, the `DropDownList` control supports the `AutoPostBack` property.

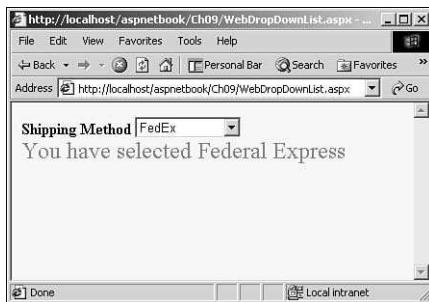
The code in Listing 9.10 illustrates how to populate a DropDownList control using ListItem controls in the HTML and respond to a selected item using a SelectedIndexChanged event handler and the AutoPostBack property.

Listing 9.10 WebDropDownList.aspx—This Page Uses ListItem Controls to Populate a DropDownList Control

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub DisplayItem(Src as Object, E as EventArgs)
    If drpShip.SelectedItem.Text <> "(please select)" Then
        lblMsg.Text = "You have selected " & _
            drpShip.SelectedItem.Value & "."
    End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <b>Shipping Method</b>
    <asp:DropDownList id="drpShip" runat="server"
        AutoPostBack=True OnSelectedIndexChanged="DisplayItem">
        <asp:ListItem>(please select)</asp:ListItem>
        <asp:ListItem
            Value="United Parcel Service">UPS</asp:ListItem>
        <asp:ListItem
            Value="Federal Express">FedEx</asp:ListItem>
        <asp:ListItem
            Value="Airborne Express">Airborne</asp:ListItem>
        <asp:ListItem
            Value="United States Postal Service">USPS</asp:ListItem>
    </asp:DropDownList>
    <br />
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

The WebDropDownList.aspx page is shown in Figure 9.5.

**Figure 9.5**

This DropDownList control lets users choose from a list of shipping methods.

The code in Listing 9.11, from WebDropDownList2.aspx, illustrates how to populate the drpShip DropDownList control using the `Items.Add` method.

Listing 9.11 WebDropDownList2.aspx—This Code Populates a DropDownList Control Using the Items.Add Method

```
Sub Page_Load(Src as Object, E as EventArgs)
If Not Page.IsPostBack Then
    drpShip.Items.Add(New ListItem("UPS", "United Parcel Service"))
    drpShip.Items.Add(New ListItem("FedEx", "Federal Express"))
    drpShip.Items.Add(New ListItem("Airborne", "Airborne Express"))
    drpShip.Items.Add(New ListItem("USPS", _
        "United States Postal Service"))
End If
End Sub
```

ListBox

While not as compact in its appearance, the ListBox server control behaves similarly to the DropDownList server control. Unlike the DropDownList control, however, you can set the ListBox server control's `SelectionMode` property to `Multiple` to allow multiple selections. The default value of `SelectionMode` is `Single`.

TIP

When you set the value of `SelectionMode` from an HTML attribute, you set it to one of the following strings: `Single` or `Multiple`. When you set the value of `SelectionMode` from code, however, you must set the value of `SelectionMode` to one of the `ListSelectionMode` enumeration values:

`ListSelectionMode.Single` or `ListSelectionMode.Multiple`.

To ascertain which items are selected in a multiple-selection ListBox control, you need to iterate through the `Items` collection and check the `Selected` property of each item. This technique is illustrated in WebListBox.aspx found in Listing 9.12.

Listing 9.12 WebListBox.aspx—Iterating through a ListBox Control's Items Collection

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
    If Not Page.IsPostBack Then
        lstItems.Items.Add(New ListItem("Apples"))
        lstItems.Items.Add(New ListItem("Cherries"))
        lstItems.Items.Add(New ListItem("Peaches"))
        lstItems.Items.Add(New ListItem("Plums"))
    End If
End Sub
Sub DisplayItem(Src as Object, E as EventArgs)
    Dim itm As ListItem
    lblMsg.Text = ""

    ' Iterate through the items collection
    ' of the list box to determine which
    ' items have been selected.
    For Each itm In lstItems.Items
        If itm.Selected Then
            lblMsg.Text &= itm.Value & ", "
        End If
    Next

    If lblMsg.Text.length > 0 Then
        lblMsg.Text = "Selected items: " & _
        lblMsg.Text.Substring(0,lblMsg.Text.Length-2) & _
        "."
    Else
        lblMsg.Text = "No items were selected."
    End If
End Sub
</script>
</head>

<body>
<form runat="server">
    <b>Groceries</b><br />
    <asp:ListBox id="lstItems" runat="server" 
        Rows=4 SelectionMode="Multiple"
        OnSelectedIndexChanged="DisplayItem" />
    <br />
```

Listing 9.12 continued

```
<asp:Button id="cmdSubmit" runat="server"
    text="Select"/>
<br />
<asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

The WebListBox.aspx page is shown in Figure 9.6.

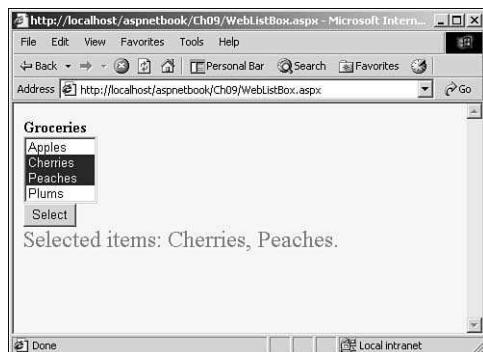


Figure 9.6

This ListBox control supports multiple selections.

Button

You can use the Button server control on a Web form to create either a submit-style or command button-style control. To create a submit-style Button control, you attach an event handler to the Click event using the OnClick attribute of the Button control. This works best when you are working with only one submit button for the form. For example, the code shown in Listing 9.13 illustrates the use of the OnClick attribute to handle Button click events.

Listing 9.13 WebButton.aspx—This Page Uses a Button Control's Click Event to Handle the Posting of the Form

```
<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub SayHello(Src as Object, E as EventArgs)
    lblMsg.Text = "Hello " & txtName.Text
End Sub
</script>
```

Listing 9.13 continued

```

End Sub
</script>
</head>

<body>
<form runat="server">
    Name:
    <asp:Textbox id="txtName" runat="server"
        TextMode="Singleline" />
    <asp:Button id="cmdPost" runat="server"
        Text="Ok" OnClick="SayHello" />
    <br />
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>

```

To create a command button-style control, you attach an event handler to the Command event using the OnCommand attribute of the Button control. This is useful when you wish to have multiple command buttons in a form. When using the OnCommand attribute, you can use the CommandName and CommandArgument attributes to pass additional information to the OnCommand event handler. The code in Listing 9.14 contains two Button server controls, cmdOk and cmdCancel. This example makes use of the CommandName attribute but not the CommandArgument attribute.

Listing 9.14 WebButton2.aspx—This Page Uses Two Button Server Controls and the Command Event

```

<html>
<head>
<style>
    span {font-size: 18pt; color: red;}
    body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub ProcessBtn(Src as Object, E as CommandEventArgs)
    If E.CommandName = "Ok" Then
        lblMsg.Text = "Welcome " & txtName.Text & "." & _
            "Thank you for your order."
    Else
        lblMsg.Text = "Your order has been cancelled!"
    End If
End Sub
</script>
</head>

<body>

```

Listing 9.14 continued

```
<form runat="server">
    Name:
    <asp:Textbox id="txtName" runat="server"
        TextMode="Singleline" />
    <asp:Button id="cmdOk" runat="server"
        Text="Ok" OnCommand="ProcessBtn"
        CommandName="Ok" />
    <asp:Button id="cmdCancel" runat="server"
        Text="Cancel" OnCommand="ProcessBtn"
        CommandName="Cancel" />
    <br />
    <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

The WebButton2.aspx page is shown in Figure 9.7.

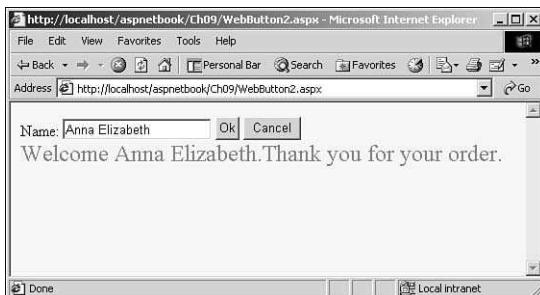


Figure 9.7

The Command event can be used to handle clicks for multiple Button controls.

ImageButton

Using the `ImageButton` server control, you can create an image that handles click events. Like the `Image` control, you set the source of the image for the `ImageButton` control using the `ImageUrl` property. The `ImageButton` control also supports the `AlternateText` and `ImageAlign` properties.

Like the `Button` control, you can handle click events for the `ImageButton` control using either the `OnClick` or `OnCommand` attributes. The `OnClick` attribute works best when you need to handle the click event of a single `ImageButton` control or when you need to determine the location at which the image was clicked. For example, the code shown in Listing 9.15 illustrates the use of the `OnClick` attribute to handle `ImageButton` click events.

Listing 9.15 WebImageBtn.aspx—The ImageClickEventArgs Parameter Supplies X and Y Coordinate Information to the OnClick Event Handler for ImageButton Controls

```
<%@ Page Language="vb" %>
<html>
<head>
<style>
    span {font-size: 18pt; color: blue;}
    span.coords {font-size: 12pt; color: red;}
    a {font-size: 18pt; }
</style>
<script language="VB" runat="server">
Sub HandleClick(Src as Object, E as ImageClickEventArgs)
    If ibtAspConnections.ImageURL = "aspconnectionslogo.gif" Then
        ibtAspConnections.ImageURL = "aspconnectionslogo2.gif"
    Else
        ibtAspConnections.ImageURL = "aspconnectionslogo.gif"
    End If
    lblCoordinates.Text = "You clicked at " & E.X & ", " & E.Y & "."
End Sub
</script>
</head>

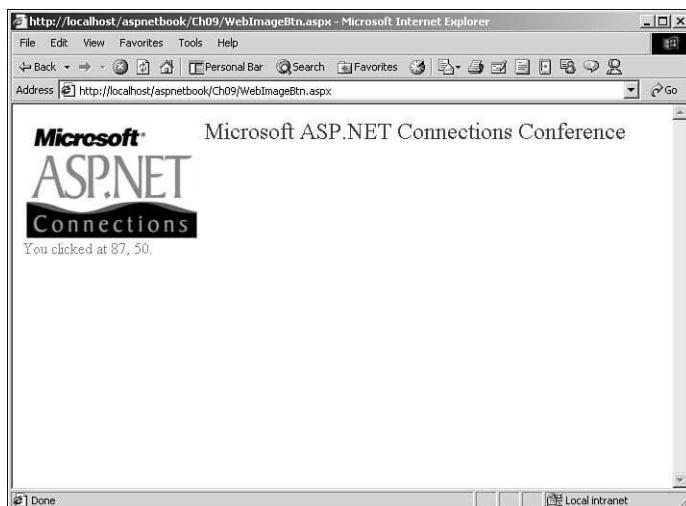
<body>
<form runat="server">
    <asp:ImageButton id="ibtAspConnections" runat="server" align="left"
        ImageUrl = "aspconnectionslogo.gif"
        AlternateText = "Micrsoft ASP.NET Connections"
        OnClick="HandleClick"
        />

    <asp:Label id="lblText" runat="server"
        text = "Microsoft ASP.NET Connections Conference" />

    <br clear="all" />

    <asp:Label id="lblCoordinates" runat="server"
        class="coords" />
</form>
</body>
</html>
```

When clicked, the code in Listing 9.15 cycles the ImageURL between aspconnectionslogo.gif and aspconnectionslogo2.gif. In addition, the lblCoordinates label control is updated with the coordinates of the mouse click by virtue of the X and Y properties passed to the event handler as part of the ImageClickEventArgs parameter. WebImageBtn.aspx is shown in action in Figure 9.8.

**Figure 9.8**

Using the `ImageButton`'s `OnClick` event handler, you can return information about the location on the image where the mouse was clicked.

The `OnCommand` attribute is best suited to handle mouse clicks for several images using a single event handler. You might use this technique, for example, to handle clicks on a site navigation bar that was made up of several images. When using the `OnCommand` attribute, you use the `CommandName` and `CommandArgument` attributes to pass additional information to the `OnCommand` event handler. This technique is illustrated in Listing 9.16.

Listing 9.16 WebImageBtn2.aspx—The `ImageButton`'s `OnCommand` Event Handler Can Handle Click Events for Multiple Images

```
<%@ Page Language="vb"%>
<html>
<head>
<style>
    span {font-size: 18pt; color: blue;}
    a {font-size: 18pt;}
</style>
<script language="VB" runat="server">
Sub HandleCmd(Src as Object, E as CommandEventArgs)
    lblText.Text = "You clicked on a " & _
        E.CommandName & " item of type " & E.CommandArgument & "."
End Sub
</script>
</head>

<body>
```

Listing 9.16 continued

```
<form runat="server">
  <asp:ImageButton id="ibtTruck" runat="server" align="left"
    ImageUrl="truck.gif"
    AlternateText="Truck"
    OnCommand="HandleCmd"
    CommandName="Transportation"
    CommandArgument="Truck"
    Width="66" Height="52"
  />
  <asp:Label id="lblText" runat="server" />
  <br clear="all" />
  <asp:ImageButton id="ibtShovel" runat="server" align="left"
    ImageUrl="shovel.gif"
    AlternateText="Shovel"
    OnCommand="HandleCmd"
    CommandName="Transportation"
    CommandArgument="Shovel"
    Width="66" Height="52"
  />
  <br clear="all" />
  <asp:ImageButton id="ibtTrain" runat="server" align="left"
    ImageUrl="train.gif"
    AlternateText="Train"
    OnCommand="HandleCmd"
    CommandName="Transportation"
    CommandArgument="Train"
    Width="66" Height="52"
  />
  <br clear="all" />
  <asp:ImageButton id="ibtWrench" runat="server" align="left"
    ImageUrl="wrench.gif"
    AlternateText="Wrench"
    OnCommand="HandleCmd"
    CommandName="Tool"
    CommandArgument="Wrench"
    Width="66" Height="52"
  />
  <br clear="all" />
  <asp:ImageButton id="ibtguitar" runat="server" align="left"
    ImageUrl="guitar.gif"
    AlternateText="Guitar"
    OnCommand="HandleCmd"
    CommandName="Music"
    CommandArgument="Guitar"
    Width="66" Height="52"
  />
```

Listing 9.16 continued

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

You can use the `CommandName` and `CommandArgument` attributes to pass any information of your choosing to the `OnCommand` event handler. The `WebImageBtn2.aspx` example (shown in Figure 9.9) uses `CommandName` to categorize the images in categories: Transportation, Tool, and Music. `CommandArgument` is used to more specifically describe the item.



Figure 9.9

A single `OnCommand` event handler manages mouse clicks for the five `ImageButton` controls.

Creating Post Away Forms

All the form examples in this chapter use the `runat="server"` attribute to create post back forms. One of the paradigm shifts in ASP.NET is the move towards more self-contained post back forms. However, sometimes you need to hook two pages together using a “post away” form. To create a post away form rather than a post back form, remove the `runat="server"` attribute from the `<form>` tag and add `method` and `action` attributes to the form.

TIP

If you add `method` and `action` attributes to a form that also contains the `runat="server"` attribute, ASP.NET will ignore the attributes and create a post back form.

For example, the WebPostAway1.aspx page shown in Listing 9.17 posts its form data to the WebPostAway2.aspx (shown in Listing 9.18).

Listing 9.17 WebPostAway1.aspx—A Post Away ASP.NET Form

```
<html>
<head>
<style>
  span {font-size: 18pt; color: red;}
  body {background-color: beige}
</style>
</script>
</head>

<body>
<form action="WebPostAway2.aspx" method="post">
  <b>Shipping Method</b>
  <asp:DropDownList id="drpShip" runat="server" >
    <asp:ListItem>(please select)</asp:ListItem>
    <asp:ListItem Value="United Parcel Service">UPS</asp:ListItem>
    <asp:ListItem Value="Federal Express">FedEx</asp:ListItem>
    <asp:ListItem Value="Airborne Express">Airborne</asp:ListItem>
    <asp:ListItem Value="United States Postal Service">USPS</asp:ListItem>
  </asp:DropDownList>
  <asp:Button id="btnOK" text="Ok" runat="server" />
  <br />
  <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

Listing 9.18 WebPostAway2.aspx—The Target of the Post Away Form

```
<html>
<head>
<style>
  span {font-size: 18pt; color: red;}
  body {background-color: beige}
</style>
<script language="VB" runat="server">
Sub Page_Load(Src as Object, E as EventArgs)
  lblMsg.Text = Request.Form("drpShip")
End Sub
</script>
</head>

<body>
```

Listing 9.18 continued

```
<form runat="server">
  <b>Your requested shipping method:</b>
  <br />
  <asp:Label id="lblMsg" runat="server" />
</form>
</body>
</html>
```

When you create a post away form, any code that is attached to post back events will not run.

Table Controls

ASP.NET contains several Web server controls for the creation of tables: the Table, TableRow, and TableCell controls. As you might expect, these server controls get mapped to HTML `<table>`, `<tr>`, `<td>`, and `<th>` elements.

The code in Listing 9.19 uses the table server controls to produce a simple table which is shown in Figure 9.10.

Listing 9.19 WebTable.aspx—This Code Produces a Four-Row-by-Two-Cell Table

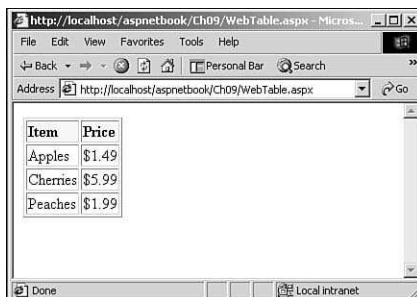
```
<html>
<body>
<asp:Table id="tbl" runat="server"
  Gridlines="both">
  <asp:TableRow BackColor="beige">
    <asp:TableCell>
      <b>Item</b>
    </asp:TableCell>
    <asp:TableCell>
      <b>Price</b>
    </asp:TableCell>
  </asp:TableRow>
  <asp:TableRow BackColor="white">
    <asp:TableCell>
      Apples
    </asp:TableCell>
    <asp:TableCell>
      $1.49
    </asp:TableCell>
  </asp:TableRow>
  <asp:TableRow BackColor="white">
    <asp:TableCell>
      Cherries
    </asp:TableCell>
```

Listing 9.19 continued

```

</asp:TableCell>
<asp:TableCell>
$5.99
</asp:TableCell>
</asp:TableRow>
<asp:TableRow BackColor="white">
<asp:TableCell>
Peaches
</asp:TableCell>
<asp:TableCell>
$1.99
</asp:TableCell>
</asp:TableRow>
</asp:Table>
</body>
</html>

```

**Figure 9.10**

A simple table produced by the table server controls.

Programmatically Generating Tables

One of the most compelling reasons to use the Web server controls to create tables is the ability to generate table cells dynamically. This is possible because the table server controls are objects. Listing 9.20 illustrates how to programmatically create table rows and cells using this technique.

Listing 9.20 WebTableGen.aspx—Generating a Table Dynamically

```

<html>
<head>
<script language="VB" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
If Page.IsPostBack Then

```

Listing 9.20 continued

```
Dim intRows As Integer
Dim intCells As Integer
Dim i As Integer = 0
Dim j As Integer = 0
Dim intRow As Integer = 0
Dim row As TableRow
Dim cell As TableCell
Dim strMsg As String

' Generate rows and cells
intRows = Convert.ToInt32(txtNumRows.Text)
intCells = Convert.ToInt32(txtNumCells.Text)

For j = 0 To intRows - 1
    ' Create a new TableRow control
    row = New TableRow()

    If (intRow Mod 2 = 0) Then
        row.BackColor = System.Drawing.Color.PowderBlue
    End If
    intRow += 1

    For i = 0 To intCells - 1
        ' Create a new TableCell control
        cell = New TableCell()
        If (i Mod 2=0) Then
            strMsg = "ASP.NET"
        Else
            strMsg = "rocks!"
        End If
        ' Add a new LiteralControl control to the
        ' the TableCell control.
        ' The LiteralControl is required to place
        ' text into the generated TableCell control
        cell.Controls.Add(New LiteralControl(strMsg))

        ' Add the cell to the Cells collection
        row.Cells.Add(cell)
    Next i

    ' Add the row to the Rows collection
    tblGen.Rows.Add(row)
```

Listing 9.20 continued

```
Next j
End If
End Sub
</script>
</head>

<body>
<form runat="server">
<font face="Verdana" size="-1">
<p>
Rows:
<asp:TextBox id="txtNumRows" runat="server"
    Columns=1 Text="1"/>
Cols:
<asp:TextBox id="txtNumCells" runat="server"
    Columns=1 Text="1"/>
<p>
<asp:button Text="Generate Table" runat="server"/>
<p>
<asp:table id="tblGen"
    CellPadding=5 CellSpacing=0
    BorderWidth=2 BorderColor="black"
    GridLines="both" runat="server" />
</font>
</form>
</body>
</html>
```

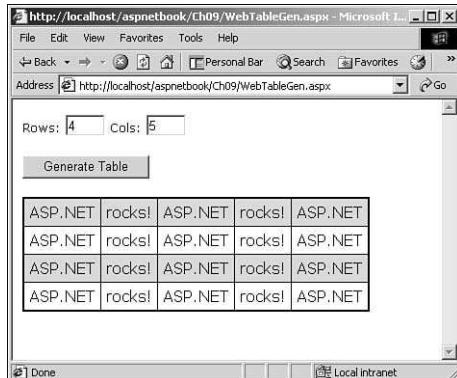
Notice how the code in Listing 9.20 creates each object in two basic steps:

1. Creates a new object using the `New` operator
2. Adds the object to the appropriate collection

In some cases, the code manipulates the new object before adding it to the collection. This is necessary, for example, when the new object is to serve as a container for other objects. In this case, the child objects are created first before adding the parent object to its collection.

In order to programmatically place text within the `TableCell` control, you must add a `LiteralControl` object to the cell to contain the text (see Listing 9.20).

The `WebTableGen.aspx` page is shown in Figure 9.11.

**Figure 9.11**

This page creates table rows and cells programmatically.

Summary

In this chapter you learned how to use Web server controls to create interactive Web forms and how Web server controls contrast with HTML server controls.

To review, the standard Web server controls can be classified into three subcategories: general controls, form controls, and table controls:

- You can use the general Web server controls to generate navigation controls, label controls, image, and panel controls.
- You can create Web forms containing various Web forms' server controls including the TextBox, Button, and DropDownList controls.
- You can dynamically create tables using the table server controls.

In the next chapter, you will explore the rich Web server controls.

CHAPTER 10

Designing Advanced Interfaces with Web Form List Controls

In the previous two chapters you learned how to use the HTML Server controls and the Web Form controls to build user interfaces with ASP.NET. In this chapter, you learn how to use a special subset of the Web Form controls, called *List controls*, to build more detailed and flexible user interfaces.

List controls allow you to add repetitive data on a page, such as a list of buttons or checkboxes, several lines of data from a database or XML file, or a complete data table in a grid. You can create sorting, paging, or even editing layouts for some of these list controls.

Basically, there are two types of list controls. The first type makes it easy to offer a collection of related checkboxes or radio buttons on a Web Form. With these controls, you need only supply a list of values, and ASP.NET will generate the set of HTML input controls.

The second set of list controls relies on templates to define the control's appearance at runtime. These templates define the "look" of the controls at runtime. For this reason, these templated controls are sometimes called "look-less" controls because they have no inherent look to them until you supply the templates.

Both the simple list controls and the templated list controls are part of the Web Forms controls family. This means that they can adjust their output based on the browser type and that they support a wide range of events and methods just like the other Web Form controls you learned about in the previous chapter.

Using Simple List Controls

The easiest list controls to implement are the checkbox and radio button list controls. These controls look and act much like the standard checkbox and radio button HTML controls. However, these controls have additional properties and events that make it easy to create a related set of checkboxes or radio buttons and then monitor click events for this related set of controls.

In the following sections, you learn how to use the checkbox and radio button list controls in your Web Forms.

The RadioButtonList Control

The RadioButtonList control allows you to create a list of radio buttons to display on a Web page. The list of buttons is automatically exclusive. This means that users can pick only one of the buttons in the list. The RadioButtonList control is a very handy control to use when you want to offer a short list of options to users in a Web Form.

For example, if you want to let users pick between three shipping options (overnight, three-day, and five-day shipping), you can use the RadioButtonList to present the options.

The code below shows a simple example of a RadioButtonList. Notice that the RadioButtonList control contains a series of `ListItem` elements just as the DropDownList control does.

Select a shipping option:

```
<asp:RadioButtonList id=rbList runat="server">
    <asp:ListItem>Overnight</asp:ListItem>
    <asp:ListItem>Three-Day</asp:ListItem>
    <asp:ListItem>Five-Day</asp:ListItem>
</asp:RadioButtonList>
```

By default, the RadioButtonList control displays the buttons vertically. However, you can use the `RepeatDirection` property to control whether it displays horizontal or vertical lists. You can also query the `SelectedItem` object to determine whether someone has set one of the buttons in the list.

The code in Listing 10.1 shows a simple ASPX page that illustrates both the `RepeatDirection` and the `SelectedItem` features of the RadioButtonList control.

Listing 10.1 RADIOPBUTTONLIST.ASPX—An Example of a Simple RadioButtonList Control

```
<%@ Page Description="RadioButtonList.aspx" %>

<script Language="VB" runat="server">

sub btnSubmit_click(sender As Object, e As EventArgs)

    if rbList.SelectedIndex > -1 then
```

Listing 10.1 continued

```
msgDisplay.Text = "You selected: " & rbList.SelectedItem.Text
else
    msgDisplay.Text = "Please select an item first!"
end if

end Sub

sub chkDirection_CheckedChanged(sender as object, e as EventArgs)
    if chkDirection.Checked=true then
        rbList.RepeatDirection = RepeatDirection.Horizontal
    else
        rbList.RepeatDirection = RepeatDirection.Vertical
    end if

end sub

</script>

<html>
<body>

<h2>Radio Button List Demo</h2>
<hr />

<form runat="server">

Select a shipping option:
<asp:RadioButtonList id=rbList runat="server">
    <asp:ListItem>Overnight</asp:ListItem>
    <asp:ListItem>Three-Day</asp:ListItem>
    <asp:ListItem>Five-Day</asp:ListItem>
</asp:RadioButtonList>

<p>
<asp:CheckBox id=chkDirection
    OnCheckedChanged="chkDirection_CheckedChanged"
    Text="Display Horizontal/Vertical"
    AutoPostBack="true"
    runat="server" />
</p>

<p>
<asp:Button id=btnSubmit Text="Submit"
    onclick="btnSubmit_Click" runat="server"/>
</p>
```

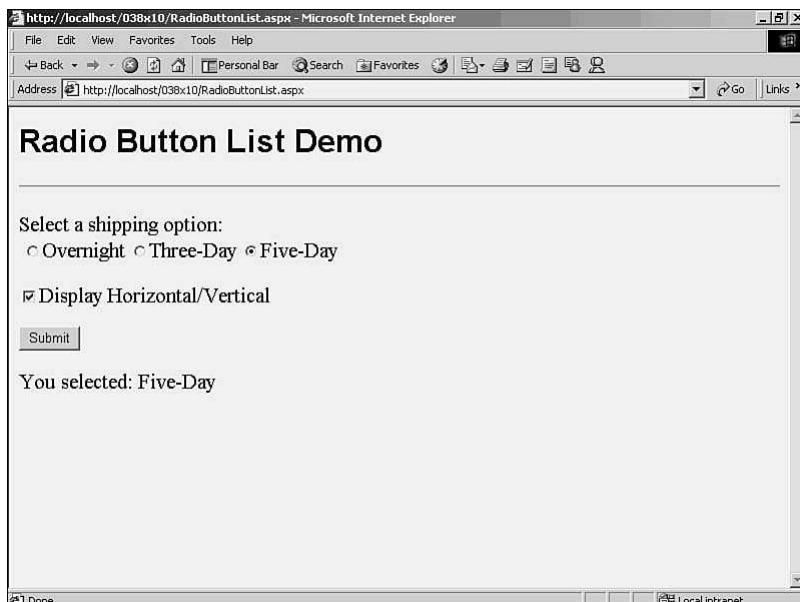
Listing 10.1 continued

```
<p>
<asp:Label id="msgDisplay" runat="server" />
</p>

</form>

</body>
</html>
```

When you load this page in your browser, you can click on the checkbox control to modify the display of the button list and select one of the buttons. Notice that you can automatically select only one of the buttons in the list. Finally, when you press the submit button on the page, the screen will be updated with the value of the item you selected (see Figure 10.1).

**Figure 10.1**

Testing the simple RadioButtonList control.

You can also use the RadioButtonList control in data bound mode. That means that you can bind an ADO.NET DataView object to the control at runtime. For example, if you had a list of options for wrapping a gift to ship to a friend, you could store these options in a DataView and then bind that view to a RadioButtonList control.

The process of binding a `DataView` to the `RadioButtonList` requires that you indicate both the display text and the associated value for the radio button. Below is a simple code example that shows how to bind a `DataView` that contains two columns: "ItemName" and "ItemAmount."

```
with rbListDB
    .DataSource = CreateDataSource()
    .DataTextField="ItemName"
    .DataValueField="ItemAmount"
    ..DataBind()
end with
```

In the code above, the `CreateDataSource()` method returns a fully populated `DataView` object. The following two lines connect the display and value properties of the radio buttons to columns in the `DataView` and the final line commits the actual binding to the data.

Once you have bound data to the control, you can access the associated text strings and values using properties of the `SelectedItem` object. This is done in the same way that you would access the items if they were added to the control declaratively (as in Listing 10.1).

Listing 10.2 shows a complete page that shows how to bind a `DataView` to the control at runtime and then retrieve the contents of both the `Text` and `Value` properties of the `SelectedItem` object.

Listing 10.2 RADIOPBUTTONLISTDB.ASPX—An Example of a Data-Bound RadioButtonList Control

```
<%@ Page Description="RadioButtonListDB.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

' display list the very first time only
sub Page_Load(Source As object, e As EventArgs)

    if IsPostBack=false then
        with rbListDB
            .DataSource = CreateDataSource()
            .DataTextField="ItemName"
            .DataValueField="ItemAmount"
            ..DataBind()
        end with
    end if

end sub

' handle clicks on the radio button
```

Listing 10.2 continued

```
sub rbListDB_Changed(sender as object, e as EventArgs)

    msgDisplay.Text = "You selected <b>" & _
        rbListDB.SelectedItem.Text & _
        "</b> with a cost of <b>" & _
        rbListDB.SelectedItem.Value & "</b>."

end sub

' create some data for this example
function CreateDataSource() as DataView

    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim i as integer

    with dt.Columns
        .Add(new DataColumn("ItemName",GetType(String)))
        .Add(new DataColumn("ItemAmount",GetType(double)))
    end with

    for i=0 to 5
        dr = dt.NewRow()
        dr(0) = "Wrapping Option " & cStr(i)
        dr(1) = 1.23 * (i+1)

        dt.rows.Add(dr)
    next

    dim dv as DataView = new DataView(dt)
    return dv
end function

</script>

<html>
<body>

<h2>Data Bound Radio Button List Demo</h2>
<hr>

<form runat="server">

Select a wrapping option for your gift:
    <asp:RadioButtonList id=rbListDB
        OnSelectedIndexChanged="rbListDB_Changed"
        AutoPostBack="true"
```

Listing 10.2 continued

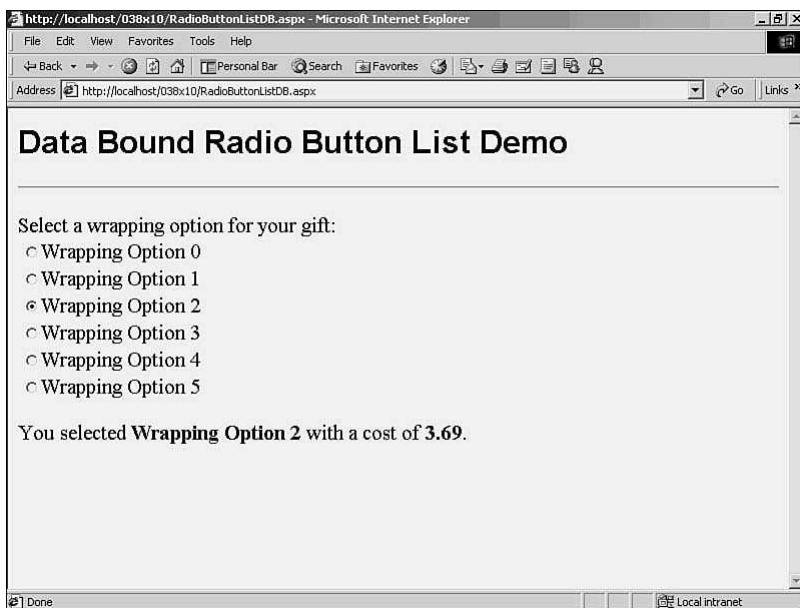
```
runat="server" />

<p>
<asp:Label id="msgDisplay" runat="server" />
</p>

</form>

</body>
</html>
```

When you load and run this page in your browser, you'll be able to select one of the many options and then see the cost of the option displayed on the page (see Figure 10.2). You should notice that the AutoPostBack property is used on the RadioButtonList control. Setting this property to "true" forces a page postback immediately after the user clicks on the button.

**Figure 10.2**

Viewing the data-bound RadioButtonList control.

The RadioButtonList control makes it very easy to create Web Forms that offer users the chance to select a single item from a list of choices. While it is possible to create this same interface using the standard HTML RadioButton control, the RadioButtonList control makes it much easier and offers the advantage of data binding.

The CheckBoxList Control

Just like there are times when you want to build a user interface that allows visitors to select a single item from a list of choices, there are also times when you want to create a Web Form that allows users to select multiple options. For example, you might want to offer users the chance to customize their weekly news feed from your site by selecting one or more categories of information. In this case, you want to allow them to check on or off a series of checkboxes to indicate their choices.

The ASP.NET CheckBoxList control offers this very option. Like the RadioButtonList control, the CheckBoxList control lets you present a list of options to the user. However, unlike the RadioButtonList control, the CheckBoxList control allows users to select more than one option in the collection.

The code below shows how you can define a CheckBoxList control in your ASP.NET page.

```
<asp:CheckBoxList id="cbList"
    RepeatColumns="3"
    RepeatDirection="Horizontal"
    TextAlign="right"
    AutoPostBack="True"
    OnSelectedIndexChanged="Check_Clicked"
    runat="server">

    <asp:ListItem>Web Site News</asp:ListItem>
    <asp:ListItem>New Products</asp:ListItem>
    <asp:ListItem>Job Postings</asp:ListItem>
    <asp:ListItem>User Newsletter</asp:ListItem>
    <asp:ListItem>Bugs and Updates</asp:ListItem>

</asp:CheckBoxList>
```

Notice that, along with the RepeatDirection attribute, you can control how many items appear in the column format and whether the prompts for the checkboxes appear to the left or right of the box itself.

When users check a box ON, this sets the Selected property of that item to “true.” You can then use server-side code to inspect the Selected property of each item to determine which ones were checked.

Listing 10.3 shows a complete page that includes code to handle the automatic post-back of the click on the checkbox and the process of inspecting the Selected property of each item in the collection.

Listing 10.3 CHECKBOXLIST.ASPX—An Example of the CheckBoxList Control

```
<%@ Page Description="CheckBoxList.aspx" %>

<script Language="VB" runat="server">

    Sub Check_Clicked(sender As Object, e As EventArgs)

        Dim i As Integer

        Message.Text = "You selected:<ul>

            For i=0 To cbList.Items.Count - 1
                If cbList.Items(i).Selected Then
                    Message.Text &lt;li> & cbList.Items(i).Text &lt;/li>
                End If
            Next

            Message.Text &lt;/ul>

    End Sub

</script>

<html>
<body>

<h2>CheckBoxList Demo</h2>
<hr />

<form runat="server">

Select all the news categories you are interested in reviewing:
<asp:CheckBoxList id="cbList"
    RepeatColumns="3"
    RepeatDirection="Horizontal"
    TextAlign="right"
    AutoPostBack="True"
    OnSelectedIndexChanged="Check_Clicked"
    runat="server">

    <asp:ListItem>Web Site News</asp:ListItem>
    <asp:ListItem>New Products</asp:ListItem>
    <asp:ListItem>Job Postings</asp:ListItem>
    <asp:ListItem>User Newsletter</asp:ListItem>


```

Listing 10.3 continued

```
<asp:ListItem>Bugs and Updates</asp:ListItem>

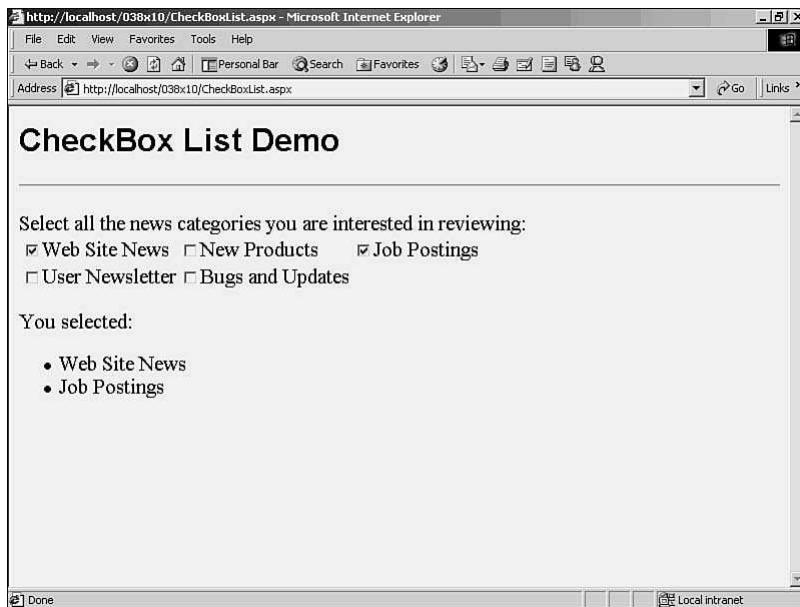
</asp:CheckBoxList>

<p>
<asp:label id="Message" runat="server" />
</p>

</form>

</body>
</html>
```

When you load and run this page, you'll be able to select any number of checkboxes and see the status of your selections in the message section of the page (see Figure 10.3).

**Figure 10.3**

Running the CheckBoxList page.

You can also bind data collections to the `CheckBoxList` control. And, like the `RadioButtonList` control, you can bind the `TextField` and the `ValueField` properties of the control to the associated text and value columns in your data collection. You can also check the `Selected` property of each item at runtime.

However, unlike the RadioButtonList control, the CheckBoxList control does not automatically update the selected status of the checkbox based on the data in the collection. To do this, you must add a bit of code to the page. This means each time you load the control you must loop through the items to check their Selected property. If it is set to “true,” you can then manually check the box ON in the user interface.

Listing 10.4 shows a complete page that binds a DataView to the CheckBoxList control. It also contains the code to inspect the Value property of the control and then set the check ON if necessary (see the Check_Load event).

Listing 10.4 CHECKBOXLISTDB.ASPX—An Example of a Data Bound CheckBoxList Control

```
<%@ Page Description="CheckBoxListDB.aspx" trace="false"%>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

' display list the very first time only
sub Page_Load(Source As object, e As EventArgs)

    if IsPostBack=false then
        with cbListDB
            .DataSource = CreateDataSource()
            .DataTextField="ItemName"
            .DataValueField="ItemSelected"
            ..DataBind()
        end with
    end if

end sub

' fix up checked display at load time
sub Check_Load(Sender As Object, E As EventArgs)

    dim i As Integer

    if Page.IsPostBack=false then
        for i = 0 to cbListDB.Items.Count-1
            if cbListDB.Items(i).Value="True" then
                cbListDB.Items(i).Selected = true
            end if
        next
        ShowResults()
    end if

end sub
```

Listing 10.4 continued

```
' build selection list for display
sub Check_Clicked(sender As Object, e As EventArgs)
    ShowResults()
end sub

' output results of checked boxes
sub ShowResults()

    Dim i As Integer

    Message.Text = "You selected:<ul>

For i=0 To cbListDB.Items.Count - 1
    if cbListDB.Items(i).Selected Then
        Message.Text &= "<li>" & cbListDB.Items(i).Text & "</li>"
    end if
Next

Message.Text &= "</ul>

end sub

' create some data for this example
function CreateDataSource() as DataView

    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim i as integer

    with dt.Columns
        .Add(new DataColumn("ItemName",GetType(String)))
        .Add(new DataColumn("ItemSelected",GetType(boolean)))
    end with

    for i=0 to 4
        dr = dt.NewRow()
        dr(0) = "Widget " & chr(i+65)
        if i=1 or i=3 then
            dr(1) = true
        else
            dr(1) = false
        end if
    next
end function
```

Listing 10.4 continued

```
dt.rows.Add(dr)
next

dim dv as DataView = new DataView(dt)
return dv

end function

</script>

<html>
<body>

<h2>Data Bound Check Box List Demo</h2>
<hr />

<form runat="server">

    <asp:CheckBoxList id="cbListDB"
        RepeatColumns="2"
        TextAlign="right"
        AutoPostBack="True"
        OnSelectedIndexChanged="Check_Clicked"
        OnLoad="Check_OnLoad"
        runat="server">

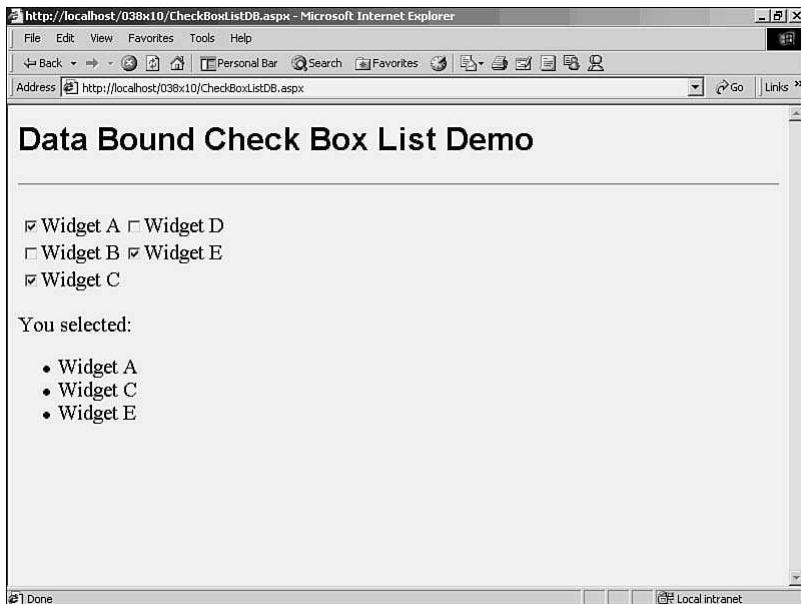
    </asp:CheckBoxList>

    <p>
        <asp:label id="Message" runat="server" />
    </p>

</form>

</body>
</html>
```

When you load and execute this page in your browser, you'll be able to select one or more items and see the updated list display at the bottom of the page (see Figure 10.4).

**Figure 10.4**

Testing the data-bound CheckBoxList page.

Now that you know how to use the simple list controls in a page, you're ready to move on to using a more complex kind of list control—the templated list control.

Using Templated List Controls

ASP.NET ships with three very powerful list controls. They are the `Repeater`, `DataList`, and `DataGrid` controls. These controls share a number of properties, methods, and events. But the most interesting thing they share is that they all support a new feature of ASP.NET: *templates*. Templates allow you to define the actual look of a control on the page instead of just accepting the default layout of the object. This gives programmers a much greater degree of control at both design and runtime.

In fact, two of the controls (`Repeater` and `DataList`) do not have any user interface at all until you add at least one template definition. Even more important, there are a number of possible templates that you can apply to a control. For example, the `Repeater` control supports five different templates. The `DataList` supports seven templates including a template for in-place editing. The `DataGrid` control supports four standard templates, plus additional column templates.

In all these examples, the templates themselves provide the user interface definitions. All three controls support a Header, Item, and Footer template. This allows you to create a display that has a customized header, footer, and item layout. You define templates by simply adding controls and HTML markup within the templates themselves.

For example, the following code shows how you can define a very simple header template for a Repeater control.

```
<HeaderTemplate>
    <h3>Product Name and Cost List</h3>
</HeaderTemplate>
```

Of course, you can also use the ASP.NET data binding service with these controls. Data binding templated controls involves the use of the DataBinder.Eval and Container.DataItem methods. For example, the code below shows how you can create an Item template that displays contents of a DataView.

```
<ItemTemplate>
    <b>
        <%#DataBinder.Eval(Container.DataItem, "ItemName")%>,
        <%#DataBinder.Eval(Container.DataItem, "ItemAmount", "{0:c}")%>
    </b>
</ItemTemplate>
```

NOTE

The details of DataBinder.Eval, and of data binding in general, are covered in Chapter 13, “Introduction to ADO.NET and Data Binding.”

In the next several sections you’ll learn how to use templated controls to create very flexible and powerful user interfaces.

The Repeater Control

The Repeater control is the most basic of the templated controls. It supports the following templates:

- **Header template**—The contents of this template are displayed once, at the start of the Repeater control.
- **Item template**—The contents of this template are displayed once for each item in the data-bound collection.
- **AlternatingItem template**—When this template is present, the contents of this template are displayed in alternation with the Item template.
- **Separator template**—This template is displayed once between each Item (or AlternatingItem) template.
- **Footer template**—The contents of this template are displayed once at the end of the Repeater control.

Listing 10.5 shows a complete page that uses each of the templates mentioned in the preceding list.

Listing 10.5 REPEATER.ASPX—Using the Repeater List Control in a Page

```
<%@ Page Description="repeater.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

sub Page_Load(sender as object, e as EventArgs)
    with dbList
        .DataSource = CreateDataSource()
        .DataBind()
    end with

end sub

' create some data for this example
function CreateDataSource() as DataView
    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim i as integer

    with dt.Columns
        .Add(new DataColumn("ItemName",GetType(String)))
        .Add(new DataColumn("ItemAmount",GetType(double)))
    end with

    for i=0 to 6
        dr = dt.NewRow()
        dr(0) = "Item" & cStr(i)
        dr(1) = 1.23 * (i+1)
        dt.rows.Add(dr)
    next

    dim dv as DataView = new DataView(dt)
    return dv
end function

</script>

<html>
<body>

<h2>Repeater Control Demo</h2>
```

Listing 10.5 continued

```
<hr />

<form runat="server">

<asp:Repeater id="dbList" runat="server">

    <HeaderTemplate>
        <h3>Product Name and Cost List</h3>
    </HeaderTemplate>

    <ItemTemplate>
        <b>
            <%#DataBinder.Eval(Container.DataItem, "ItemName")%>,
            <%#DataBinder.Eval(Container.DataItem, "ItemAmount", "{0:c}")%>
        </b>
    </ItemTemplate>

    <SeparatorTemplate>
        <br />
    </SeparatorTemplate>

    <AlternatingItemTemplate>
        <i>
            <%#DataBinder.Eval(Container.DataItem, "ItemName")%>,
            <%#DataBinder.Eval(Container.DataItem, "ItemAmount", "{0:c}")%>
        </i>
    </AlternatingItemTemplate>

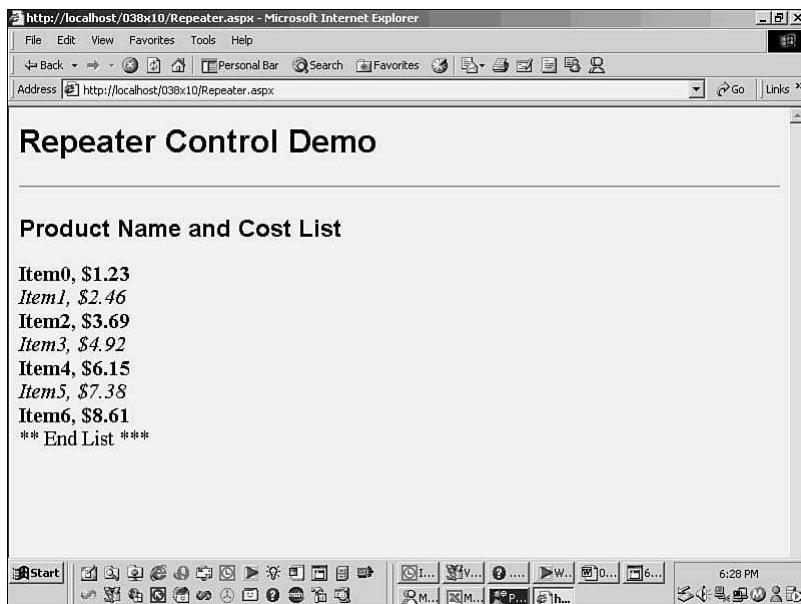
    <FooterTemplate>
        <br />
        ** End List ***
    </FooterTemplate>

</asp:Repeater>

</form>

</body>
</html>
```

As you can see in the listing, the Repeater control gives you complete control over the display of the data collection. In fact, with Repeater controls, you must define all aspects of the display yourself. When you load and run the code in Listing 10.5, you'll see a display similar to the one in Figure 10.5.

**Figure 10.5**

Viewing the Repeater list control example.

The DataList Control

While the Repeater provides programmers with full control over building the look and feel of templated controls, it has some limits. For example, you might want to create a list that tracks user selection and then presents a different template when the control is selected. Also, in some cases, you might want to create a template that defines an editable version of the display for a selected record.

The **DataList** offers such features. This control works much like the Repeater, but has more power. Also, while the Repeater offers no default HTML markup at all, the **DataList** control creates an HTML table as a wrapper around all the templates you define. This makes it easier to create well-spaced displays. It is also this table feature that allows the **DataList** to offer the **SelectedItem** and **EditItem** templates.

Following is a list of all the templates supported by the **DataList** control:

- Header Template
- Item Template
- AlternatingItem Template
- SelectedItem Template
- EditItem Template
- Separator Template
- Footer Template

The `DownList` control also supports a number of events. This means that you can register methods to handle one or more events that occur within the control. For example, the `DownList` supports the `OnItemCommand`, `OnDeleteCommand`, `OnUpdateCommand`, `OnSelectCommand`, and `OnCancelCommand` events. The code below shows an example definition for the `DownList` control that registers an event handler for the `OnItemCommand`. This event handler will be called each time the user selects an item in the list.

```
<asp:DownList id="dbList"
    BorderWidth="2"
    Width="250"

    HeaderStyle-BackColor="gray"
    AlternatingItemStyle-BackColor="silver"
    EditItemStyle-BackColor="gainsboro"

    OnItemCommand="dbList_ItemCommand"

    runat="server">
```

Notice also that the control supports a number of CSS-style properties. This allows you to directly control the color and several other style properties of each of the templates. Listing 10.6 shows a complete page that implements the `Header`, `Footer`, `Item`, and `SelectedItem` templates for a `DownList` control. It also provides an event handler for the `OnItemCommand`.

Listing 10.6 DATALIST.ASPX—A Page That Uses the `DownList` Control

```
<%@ Page Description="DownList.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

    ' shared objects
    dim Cart As DataTable
    dim CartView As DataView

    ' handles cart state management
    Sub Page_Load(sender As Object, e As EventArgs)

        CreateCart()
        CartView = New DataView(Cart)
        CartView.Sort="Item"

        if Not IsPostBack Then
            ' need to load this data only once
            BindList
    End Sub
</script>
```

Listing 10.6 continued

```
End If

End Sub

' bind cart to control
Sub BindList()

    dbList.DataSource= CartView
    dbList.DataBind

End Sub

Sub dbList_ItemCommand(sender as object, _
    args as DataListCommandEventArgs)

    dbList.SelectedIndex = args.Item.ItemIndex
    BindList()

End Sub

Sub reset_Click(sender as object, args as EventArgs)

    dbList.SelectedIndex = -1
    BindList()

End Sub

' create some data for this example
Function CreateCart()

    Dim dr As DataRow
    Dim i As Integer

    ' create a new data table object
    Cart = new DataTable()
    With Cart.Columns
        .Add(new DataColumn("Qty", GetType(String)))
        .Add(new DataColumn("Item", GetType(String)))
        .Add(new DataColumn("Price", GetType(String)))
    End With

    ' add data to table
    For i = 1 To 4
        dr = Cart.NewRow()
        If (i Mod 2 <> 0) Then
            dr(0) = "2"
        Else
            dr(0) = "1"
        End If
        dr(1) = "Item" & i
        dr(2) = "10.00"
        Cart.Rows.Add(dr)
    Next
End Function
```

Listing 10.6 continued

```
dr(0) = "1"
end if

dr(1) = "Widget " & chr(i+64)
dr(2) = (1.23 * (i + 1)).ToString

Cart.Rows.Add(dr)
Next

return Cart

end function

</script>

<html>
<body>

<h2>DataList Demo</h2>
<hr />

<form runat="server" color=>

    <asp:DataList id="dbList"
        BorderWidth="2"
        Width="250"

        HeaderStyle-BackColor="gray"
        AlternatingItemStyle-BackColor="silver"
        EditItemStyle-BackColor="gainsboro"

        OnItemCommand="dbList_ItemCommand"
        runat="server">

        <HeaderTemplate>
            <h3 align="center" valign="middle" >
                Product List &nbsp;&nbsp;&nbsp;
                <asp:Button Text="Reset" OnClick="reset_click" runat="server"/>
            </h3>
        </HeaderTemplate>

        <ItemTemplate>
            <asp:LinkButton id="btnEdit" runat="server"
                Text='<%# Container.DataItem("Item")%>'
                CommandName="Select" />
        </ItemTemplate>

        <SelectedItemTemplate>
```

Listing 10.6 continued

```
<table>
    <tr>
        <td>Item:</td>
        <td><asp:Label id="item" runat="server"
            Text='<%# Container.DataItem("Item") %>' />
        </td>
    </tr>
    <tr>
        <td>Quantity:</td>
        <td><asp:TextBox id="qty" size="15" runat="server"
            Text='<%# Container.DataItem("Qty") %>' />
        </td>
    </tr>
    <tr>
        <td>Price:</td>
        <td><asp:TextBox id="price" size="15" runat="server"
            Text='<%# DataBinder.Eval(Container.DataItem, _
                "Price") %>' />
        </td>
    </tr>
</table>
</SelectedItemTemplate>

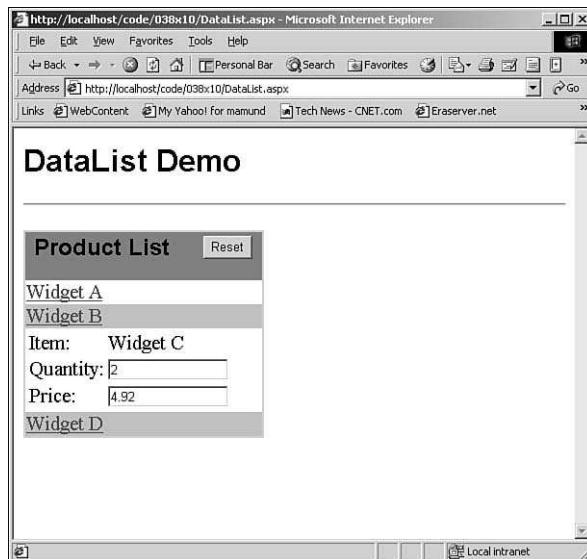
</asp:DataList>

</form>

</body>
</html>
```

In the example shown in Listing 10.6, the SelectedItem template is defined to include input controls. This allows users to select a row in the list and then edit the detail contents of that row. This is done by manipulating the `SelectedIndex` property of the `DataList`. For example, you will see that the Reset button that appears in the Header template merely sets the `SelectedIndex` property to -1, meaning that there is no item selected.

When you load and run this page in your browser and select one of the rows, you'll see the input controls ready for updates.

**Figure 10.6**

Testing the SelectedItem template of a `DataList` control.

The `DataGridView` Control

The most powerful of the ASP.NET list controls is the `DataGridView` control. This control supports more than just data binding and templates. It also supports sorting, paging, and in-place editing. Another very handy aspect of the `DataGridView` control is that, unlike the `Repeater` and `DataList`, the `DataGridView` has a default user interface. That means that you can display bound data in the `DataGridView` control without having to define any templates at all.

The simplest way to use the `DataGridView` control is to just bind a `DataView` to the control at runtime. Listing 10.7 shows a complete page that does this.

Listing 10.7 DATAGRID.ASPX—A Simple Page That Shows a Data-Bound DataGridView Control

```
<%@ Page Description="DataGridView.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

' display list the very first time only
sub Page_Load(Source As object, e As EventArgs)
    dgList.DataSource = CreateDataSource()
    dgList.DataBind
end sub

```

Listing 10.7 continued

```
end sub

' create some data for this example
function CreateDataSource() as DataView

    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim i as integer

    with dt.Columns
        .Add(new DataColumn("ItemName",GetType(String)))
        .Add(new DataColumn("ItemAmount",GetType(double)))
        .Add(new DataColumn("StatusFlag",GetType(boolean)))
    end with

    for i=0 to 15
        dr = dt.NewRow()
        dr(0) = "Item" & cStr(i)
        dr(1) = 1.25 * (i+1)
        If (i Mod 2 <> 0) Then
            dr(2) = True
        Else
            dr(2) = False
        End If

        dt.rows.Add(dr)
    next

    return dt.DefaultView
end function

</script>

<html>
<body>

<h2>DataGrid Sorting Demo</h2>
<hr />

<form runat="server">

    <asp:DataGrid id=dgList runat="server" />

</form>

</body>
</html>
```

Notice that, without the code needed to create that DataView itself, this page would be quite small. All that is needed is the declaration of the DataGrid control in the markup portion of the page and the data binding code in the Page_Load event of the page.

This is a very simple example of how you can quickly create a data display using the DataGrid control. In the next several sections, you'll learn how you can add sorting, paging, and editing to your DataGrid pages.

Sorting DataGrid Contents

At times, you might want to allow users to sort the data within a grid display. This can be done relatively easily with the DataGrid control. All you need to do is set the AllowSorting attribute to "true" and to register a sorting method for the control with the OnSortCommand attribute. The only serious work that needs to be done is the actual sorting of the data within the event handler you registered with the control. The following code example shows what the event handler for the OnSortCommand looks like:

```
Sub dgList_Sort(sender As Object, e As DataGridSortCommandEventArg)
    SortField = e.SortExpression
    BindGrid
End Sub
```

Luckily, the DataView has a Sort property. All you need to do is set the Sort property of the DataView to the name of the column you wish to use for sorting. Then, after populating the DataView, you can simply set the Sort property, and the data will be displayed in sorted order.

The code in Listing 10.8 shows a complete page that implements the sorting feature of the DataGrid control.

Listing 10.8 DATAGRIDSORT.ASPX—Implementing the Sort Feature of the DataGrid Control

```
<%@ Page Description="DataGridSort.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

dim sortField as string

' display list the very first time only
sub Page_Load(Source As object, e As EventArgs)
    BindGrid
end sub

Sub dgList_Sort(sender As Object, e As DataGridSortCommandEventArg)
    SortField = e.SortExpression
    BindGrid
End Sub

Sub BindGrid()
```

Listing 10.8 continued

```
dgList.DataSource = CreateDataSource()
dgList.DataBind
End Sub

' create some data for this example
function CreateDataSource() as DataView

    dim dt as DataTable = new DataTable()
    dim dr as DataRow
    dim i as integer

    with dt.Columns
        .Add(new DataColumn("ItemName",GetType(String)))
        .Add(new DataColumn("ItemAmount",GetType(double)))
        .Add(new DataColumn("StatusFlag",GetType(boolean)))
    end with

    for i=0 to 25
        dr = dt.NewRow()
        dr(0) = "Item" & cStr(i)
        dr(1) = 1.25 * (i+1)
        If (i Mod 2 <> 0) Then
            dr(2) = True
        Else
            dr(2) = False
        End If

        dt.rows.Add(dr)
    next

    dim dv as DataView = new DataView(dt)

    if sortField="" then
        sortField="ItemName"
    end if
    dv.Sort=sortField

    return dv

end function

</script>

<html>
<body>

<h2>DataGrid Sorting Demo</h2>
```

Listing 10.8 continued

```
<hr />

<form runat="server">

    <asp:DataGrid id=dgList
        AllowSorting="true"
        OnSortCommand="dgList_Sort"
        runat="server"/>

</form>

</body>
</html>
```

Paging DataGrids

Just as DataGrids support sorting, they also support paging. This means you can control the presentation of long lists of data by simply offering them up in a series of pages that users can select at runtime. Implementing paging is done much the same way as sorting. There is an `AllowPaging` attribute and an `OnPageIndexChanged` event that requires an event handler. You can also control the style and location of the paging buttons or links within the grid. For example, the next code snippet shows the markup declaration for a control that supports right-aligned “Next” and “Prev” buttons at the bottom of the grid.

```
<asp:DataGrid id=dgList
    PagerStyle-HorizontalAlign="Right"
    PagerStyle-NextPageText="Next"
    PagerStyle-PrevPageText="Prev"
    AllowPaging="true"
    OnPageIndexChanged="dgList_Page"
    runat="server"/>
```

You can also display a series of page numbers instead of the usual “Next” and “Prev” buttons. Regardless of the page links you display, the contents of the event handler are the same. You simply retrieve the page index from the argument list and set that page index for the DataGrid control. The following code example shows how this is done.

```
Sub dgList_Page(sender As Object, e As DataGridPageChangedEventArgs)
    dgList.CurrentPageIndex = e.NewPageIndex
    BindGrid
End Sub
```

To put it all together, Listing 10.9 shows a complete page that supports paging with the DataGrid control.

Listing 10.9 DATAGRIDPAGE.ASPX—Implementing Paging for the DataGrid Control

```
<%@ Page Description="DataGridPage.aspx" %>
<%@ import namespace="System.Data" %>

<script Language="VB" runat="server">

    ' display list the very first time only
    sub Page_Load(Source As object, e As EventArgs)
        BindGrid
    end sub

    Sub dgList_Page(sender As Object, e As DataGridPageChangedEventArgs)
        dgList.CurrentPageIndex = e.NewPageIndex
        BindGrid
    End Sub

    Sub BindGrid()
        dgList.DataSource = CreateDataSource()
        dgList.DataBind
    End Sub

    ' create some data for this example
    function CreateDataSource() as DataView

        dim dt as DataTable = new DataTable()
        dim dr as DataRow
        dim i as integer

        with dt.Columns
            .Add(new DataColumn("ItemName",GetType(String)))
            .Add(new DataColumn("ItemAmount",GetType(double)))
            .Add(new DataColumn("StatusFlag",GetType(boolean)))
        end with

        for i=0 to 99
            dr = dt.NewRow()
            dr(0) = "Item" & cStr(i)
            dr(1) = 1.25 * (i+1)
            If (i Mod 2 <> 0) Then
                dr(2) = True
            Else
                dr(2) = False
            End If
        next
    End Function

```

Listing 10.9 continued

```
End If

dt.rows.Add(dr)
next

dim dv as DataView = new DataView(dt)

return dv

end function

</script>

<html>
<body>

<h2>DataGrid Paging Demo</h2>
<hr />

<form runat="server">

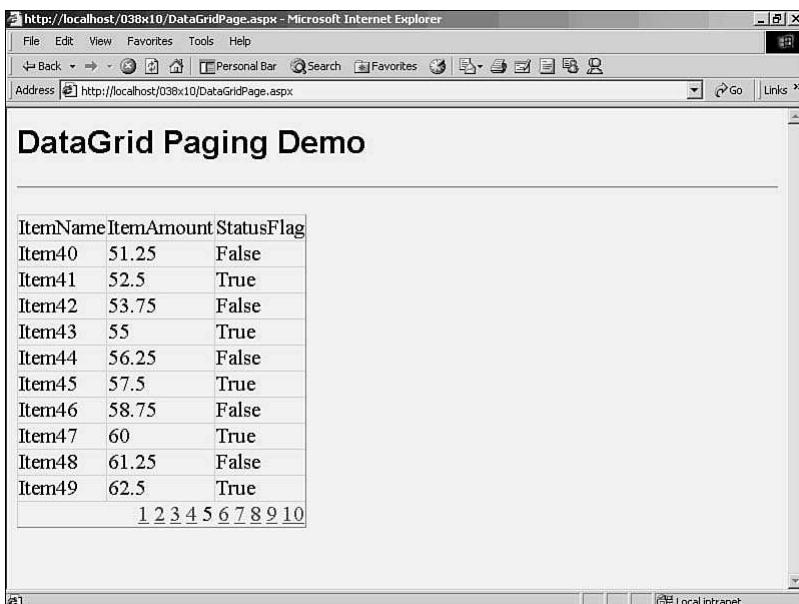
<asp:DataGrid id=dgList
    PagerStyle-Mode="NumericPages"
    PagerStyle-HorizontalAlignment="Right"
    PagerStyle-NextPageText="Next"
    PagerStyle-PrevPageText="Prev"
    AllowPaging="true"
    OnPageIndexChanged="dgList_Page"
    runat="server"/>

</form>

</body>
</html>
```

Like the sorting example shown earlier, the important code takes place in the `Page_Load` event and the event handler for the paging event. The code needed to create the `DataView` object could easily be replaced by calls to a database or to static XML files.

When you load and run this page in your browser, you'll see a series of page numbers at the bottom of the grid. Clicking on any of these page numbers forces the page to refresh and fills the grid with the contents of the desired page (see Figure 10.7).

**Figure 10.7**

Testing the paging feature of the DataGrid control.

Editing DataGrid Columns

The final feature of the DataGrid to explore in this chapter is the ability to perform in-place editing. The DataGrid supports a number of special templates, much like the DataList and Repeater controls. However, unlike the DataList and Repeater, the DataGrid supports templated columns themselves. This means that you can exercise a great deal of control over how each column is displayed within the DataGrid. In fact, you can even control which columns appear at all.

This is done through the use of a `TemplateColumn` element within the DataGrid. `TemplateColumn` elements allow you to control both the heading and the contents of each column in each row of the display. For example, the code below shows a template for the “Quantity” column that defines the display of a data-bound text box for each row of the grid.

```
<asp:TemplateColumn HeaderText="Quantity">
    <ItemTemplate>
        <asp:TextBox id=txtQty runat="server"
            Text='<%# DataBinder.Eval(Container.DataItem, "Qty") %>'
            Width="40px"
        />
    </ItemTemplate>
</asp:TemplateColumn>
```

You can also include a simple definition of a data-bound column within a `DataGridView` using the `BoundColumn` element. The code below shows how you can define a column with the heading “Price” that displays the “Price” column from the `DataView` and formats the output as currency.

```
<asp:BoundColumn
    HeaderText="Price"
    DataField="Price"
    DataFormatString="{0:c}"
/>
```

Both bound columns and template columns are wrapped in a single `Columns` element. This defines the set of column definitions for the `DataGridView`. Listing 10.10 shows a complete page that implements the in-place editing within a `DataGridView`.

Listing 10.10 DATAGRIDEDIT.ASPX—Implementing In-Place Editing within a DataGridView Control

```
<%@ Page Description="editing in a data grid"%>
<%@ import namespace="System.Data" %>

<script language="VB" runat="server">

Dim runningTotal As Double = 0

sub Page_Load(sender As Object, e As EventArgs)
    if not IsPostBack then
        bindGrid
    end if
end sub

sub btnUpdate_click(sender As Object, e As EventArgs)
    dim i as integer
    dim dr As DataRow

    ' move new qty values into dataset
    for i = 0 To dbList.Items.Count - 1
        dr = Cart.Rows(i)
        dr(0) = CType(dbList.items(i).FindControl("txtQty"), TextBox).Text
    next
    bindGrid
end sub

sub bindGrid()
    dbList.DataSource = Cart.DefaultView
    dbList.DataBind()
end sub

function getTotal (count As Integer, price As Double) As Double
```

Listing 10.10 continued

```
dim total As Double

' update total for item and cart
total = count * price
runningTotal += total
grossAmount.Text=String.Format("{0:c}",runningTotal)

return(total)
end function

ReadOnly Property Cart As DataTable
get
    ' get the data from a session var
    if Session("dbList_Cart") Is Nothing Then
        dim tmpCart as DataTable = createData()
        Session("dbList_Cart")=tmpCart
    end if
    return Session("dbList_Cart")
end get
end property

function createData() as DataTable
    ' create new data and place it in a session var
    dim tmpCart as new DataTable()
    dim i As Integer
    dim dr As DataRow

    tmpCart.Columns.Add(new DataColumn("Qty", GetType(String)))
    tmpCart.Columns.Add(new DataColumn("Product", GetType(String)))
    tmpCart.Columns.Add(new DataColumn("Price", GetType(Double)))

    for i= 1 to 6
        dr = tmpCart.NewRow()
        dr(0) = "1"
        dr(1) = "Product " & i.ToString
        dr(2) = 1.25 * (i+1)
        tmpCart.Rows.Add(dr)
    next

    return tmpCart
end function

</script>
<html>
<body>
```

Listing 10.10 continued

```
<h2>DataGrid Edit Demo</h2>
<hr />

<form runat="server">

<asp:DataGrid id="dbList" runat="server"
  AutoGenerateColumns="false">
  <Columns>
    <asp:TemplateColumn HeaderText="Quantity">
      <ItemTemplate>
        <asp:TextBox id=txtQty runat="server"
          Text='<%# DataBinder.Eval(Container.DataItem, "Qty") %>'
          Width="40px"
        />
      </ItemTemplate>
    </asp:TemplateColumn>
    <asp:BoundColumn HeaderText="Product" DataField="Product"/>
    <asp:BoundColumn HeaderText="Price" DataField="Price"
      DataFormatString="{0:c}" />
    <asp:TemplateColumn HeaderText="SubTotal">
      <ItemTemplate>
        <p align="right">
          <asp:Label runat="server"
            Text='<%# System.String.Format("{0:c}", _
              getTotal(Int32.Parse(Container.DataItem("Qty")), _ 
              Container.DataItem("Price")))%>'>
        />
        </p>
      </ItemTemplate>
    </asp:TemplateColumn>
  </Columns>
</asp:DataGrid>

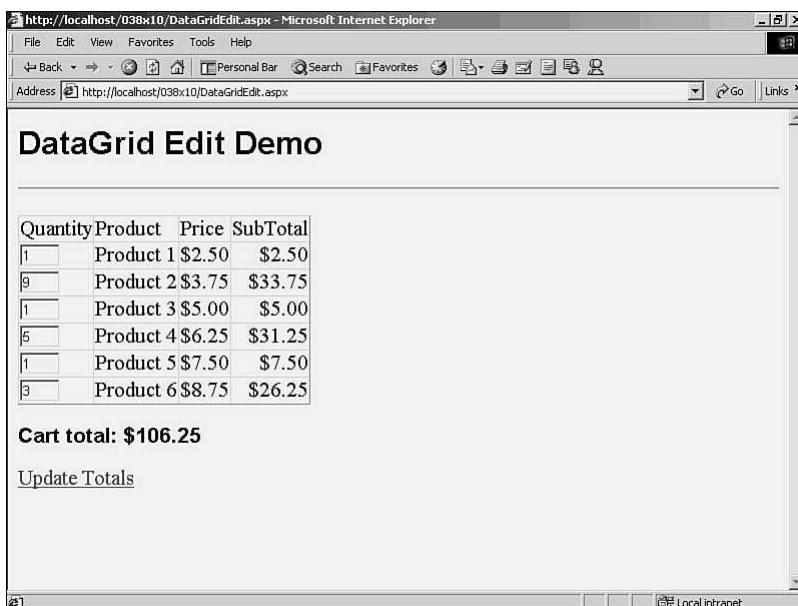
<h4>
Cart total:
<asp:Label id="grossAmount" runat="server"/>
</h4>

<asp:LinkButton id=btnUpdate runat="server"
  Text="Update Totals"
  onClick="btnUpdate_click"/>

</form>

</body>
</html>
```

You'll also notice that this example shows how to bind to a function within a data grid column and provides a grand total for all the amounts in the data grid. When you load and run this page in your browser, you'll be able to update the item counts within the grid and then press the "Update Totals" link to recalculate the final amounts (see Figure 10.8).

**Figure 10.8**

Testing the *in-place* editing feature of the DataGrid.

Summary

In this chapter, you learned how to use a special subset of the Web Form controls, called List controls, to build more detailed and flexible user interfaces.

You learned that there are two types of list controls: simple list controls and templated list controls. The first type includes the RadioButtonList and CheckBoxList controls. These controls make it easy to offer a collection of related checkboxes or radio buttons on a Web Form.

The second set of list controls includes the Repeater, DataList, and DataGrid controls. This set of controls relies on templates to describe their appearance at runtime. These templates define the look of the controls at runtime. For this reason, these templated controls are sometimes called "look-less" controls because they have no inherent look to them until you supply the templates.

CHAPTER 11

Improving Your User Interfaces with Validation Controls

In Chapter 9, “Creating Interactive Forms with Web Server Controls,” you learned how to create Web forms using the Web controls built into the Microsoft .NET framework. This chapter examines a special type of Web control called *validation controls*. Validation controls give you control of the data that the users of your applications submit—in most cases, without the need to write a single line of client-side script! After reading this chapter, you will understand how to use validation controls to make your applications more robust and user-friendly.

The chapter begins with a look at validation controls and how they fit into the Microsoft .NET framework. Then it examines the simple validation controls, including `RequiredFieldValidator`, `CompareValidator` and `RangeValidator`. The chapter concludes with a look at some of the more difficult validation controls, including the `RegularExpressionValidator`, `CustomValidator` and the `ValidationSummary` controls.

What Are Validation Controls

As their name suggests, validation controls are primarily used to validate, or verify, that the data a user enters into a Web form is correct. In ASP “classic” this could be accomplished in a number of different ways. Many developers used client-side languages such as VBScript and JavaScript to perform these operations on the client side. This was not always an optimal solution, or even possible at all if users had client-side scripting disabled in their browsers. Server-side script was also used to validate user input, but this required that the browser make an unnecessary round trip to the server.

Validation controls attempt to fix the existing problems with the validation model. Because they exist as server-side objects just like any other Web control, they are easily instantiated, as in the following example:

```
<asp:RequiredFieldValidator runat="server"
    ErrorMessage="You must enter a value"
    ControlToValidate="txtInput">
</asp:RequiredFieldValidator>
```

When you place this validation control on a Web form, the user must enter something into the `txtInput` textbox in order to successfully submit the form. If the user attempts to submit the form with the field left blank, the error message “You must enter a value” appears. (You’ll see more about the `RequiredFieldValidator` control in the next few sections.)

Though validation controls are server-side objects, they can generate the client-side script necessary to notify the user of an error immediately after they submit the form, before the browser contacts the server. Not only does this improve the appearance of the application to the end user, but it also reduces the total number of HTTP requests sent to the Web server. This can have a dramatic effect on the performance of your application. Validation controls will only generate client-side script for browsers that support DHTML (Dynamic HTML), such as Microsoft Internet Explorer versions 4.0 and higher. You can set validation controls to never create client-side script by setting the `EnableClientScript` property of the validation control to `False`.

Properties and Methods Common to All Validation Controls

Each of the five individual validation controls derives from both the `WebControl` and `BaseValidator` base classes. This ensures a consistent object model and functionality common to all validation controls. The base classes can be found in the `System.Web.UI.WebControls` namespace.

Because the validation controls derive from the `BaseValidator` class, there are a number of common properties. Each control has a property named `ControlToValidate`. This property contains the name of the Web control on which you wish to perform validation.

The next common property is `Display`. The `Display` property controls how a validation control appears on the page. This property can be set to one of three different values: `Dynamic`, `Static`, or `None`. If set to `Dynamic`, the validation control only takes up space when displaying an error. Unless you carefully design your HTML with this in mind, you’ll notice that the HTML on your page may noticeably shift when an error is generated. Setting this property to `Static` generates the opposite behavior: The validation control takes up the same amount of space regardless of whether the error text is displayed. Finally, if set to `None`, the validation control is never visible to the user. The `None` setting makes sense when using the `ValidationSummary` control. You can use the `ValidationSummary` control to display a collective error message from a number of individual controls, each with `Display` set to `None`.

Another property common to all validation controls is `ErrorMessage`. The `ErrorMessage` property contains the text you would like to display to the user if an error occurs. You must enter an error message for each control, because no default error messages exist. It's also important to note that `ErrorMessage` can contain HTML instead of just plain text. This can be used to add extra formatting information to the error text.

NOTE

Validation control error messages appear exactly where the validation control is located in your Web form. Therefore, the placement of your validation controls is important. A few logical choices include placing them at the top of your page, or next to the control they are validating, to visually highlight the error for the user.

Finally, the `IsValid` property contains the validation control's current validation state. If no errors have been detected in the user's input, `IsValid` will be set to `True`. This should not be confused with the `Page.IsValid` property. `Page.IsValid` enables you to programmatically check whether there have been any validation errors on the entire Web form.

TIP

If you check `Page.IsValid` from the `Page_Load` event, you will find that it's not available; `Page.IsValid` is set by ASP.NET after `Page_Load` occurs. If you wish to check the value of `Page.IsValid`, you need to check it from an event procedure that occurs later in the page lifecycle, such as during the `Page_PreRender` event or an event of a control.

Simple Validation Controls

The validation controls in the Microsoft .NET framework can be loosely organized into two groups: simple and advanced. In this first section, the simpler validation controls are covered. You learn how to use the `RequiredFieldValidator` control to make sure a user does not leave any required fields empty. You also encounter the `CompareValidator` control and learn how to use this to easily verify that two user inputs are the same. Lastly, you learn how to use the `RangeValidator` control to ensure that a user input falls within an acceptable range of values.

The RequiredFieldValidator Control

As previously mentioned, the `RequiredFieldValidator` validation control is used to make sure the user enters some information in a field. This is probably the most commonly used of all validation controls.

Consider that requiring the user to enter something in a field is not necessarily the same thing as ensuring that a field is not left blank. Though the `RequiredFieldValidator` control can be used to ensure that a field is not left blank, it can also be used to make sure the user changes a default field value. For instance, say you have a text box for the user's phone number. You can suggest the phone number format to the user by placing some default text in the text box, such as "XXX-XXX-XXXX".

Because there is something in the field, this would enable the user to submit the form unless you specify the `InitialValue` property of the `RequiredFieldValidator`. By setting this to “XXX-XXX-XXXX”, you can ensure that the user filling out the form must change this default value in order to submit the form. By default, `InitialValue` is blank, ensuring that blank required fields are not submitted.

Listing 11.1 shows the simplest and most common use of the `RequiredFieldValidator` control. If a user attempts to submit this form without entering something in the textbox, an error message is displayed informing them of the omission.

Listing 11.1 RequiredFieldValidator.aspx—A Web Form Utilizing the RequiredFieldValidator

```
<html>
  <body>
    <form id="Form1" method="post" runat="server">
      Please enter your phone number below:
      <br />
      <asp:TextBox id="txtPhoneNumber" runat="server">
      </asp:TextBox>
      <input type="submit" runat="server" />
      <br />

      <asp:RequiredFieldValidator runat="server"
        ControlToValidate="txtPhoneNumber"
        Display="Static"
        ErrorMessage="You must enter a phone number to continue">
      </asp:RequiredFieldValidator>
    </form>
  </body>
</html>
```

The `RequiredFieldValidator.aspx` page is shown in Figure 11.1.

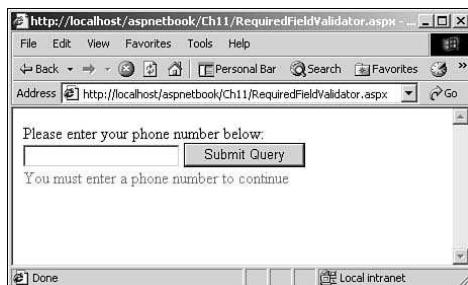


Figure 11.1

This page employs the `RequiredFieldValidator` control to require an entry in the phone number field.

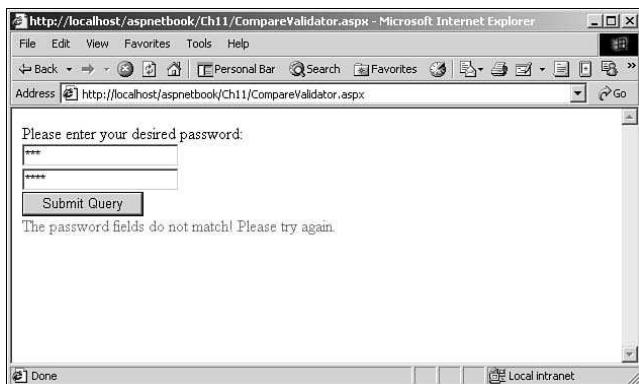
The CompareValidator Control

You can use the CompareValidator control in two ways. You can use it to make sure that the user enters identical data in two input fields. A typical and often-used case where this comes in handy is to make sure a user has entered the same password in two fields to eliminate the chance of a typing error. You can also use the CompareValidator control to compare a user input to a constant value.

In order to compare two fields, you specify the ControlToValidate property (as in any other validation control), and additionally specify the ControlToCompare property as well. This can be seen in the example in Listing 11.2. In order to submit the form, a user must enter the same value into both fields. Also notice two RequiredFieldValidators in the form. This is to ensure the user does not enter a blank password. If an input is blank, validation is not triggered.

Listing 11.2 CompareValidator.aspx—A Web Form Verifying a User Has Entered a Password Correctly

```
<html>
  <body>
    <form id="Form1" method="post" runat="server">
      Please enter your desired password:
      <br />
      <asp:TextBox runat="server"
        TextMode="Password"
        id="txtPassword">
      </asp:TextBox>
      <asp:RequiredFieldValidator runat="server"
        ControlToValidate="txtPassword"
        ErrorMessage="You must enter a password to continue!">
      </asp:RequiredFieldValidator>
      <br />
      <asp:TextBox runat="server"
        TextMode="Password"
        id="txtConfirmPassword">
      </asp:TextBox>
      <asp:RequiredFieldValidator runat="server"
        ControlToValidate="txtConfirmPassword"
        ErrorMessage="You must enter a password to continue!">
      </asp:RequiredFieldValidator>
      <br />
      <input type="submit" runat="server" />
      <br />
      <asp:CompareValidator runat="server"
        ControlToValidate="txtConfirmPassword"
        ControlToCompare="txtPassword"
        ErrorMessage="The password fields do not match! Please try again.">
      </asp:CompareValidator>
    </form>
  </body>
</html>
```

**Figure 11.2**

This page compares the values of the two password controls using the CompareValidator control.

In order to compare a user input to a constant value, you simply specify a value in the ValueToCompare property. The control will be compared against this value. The ControlToCompare and ValueToCompare properties are mutually exclusive; you can only use one or the other for a particular CompareValidator.

The RangeValidator Control

As its name implies, you use the RangeValidator control to ensure that a user enters a value that falls into a range of acceptable values. The maximum and minimum user inputs are specified by the MaximumValue and MinimumValue properties, respectively. Listing 11.3 demonstrates how to use the RangeValidator control to make sure a user enters a value between 1 and 10.

Listing 11.3 RangeValidator.aspx—A Web Form Verifying a User Has Entered a Number between 1 and 10

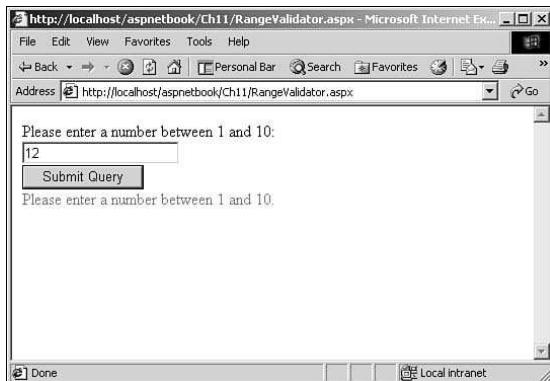
```
<html>
  <body>
    <form id="Form1" method="post" runat="server">
      Please enter a number between 1 and 10:
      <br>
      <asp:TextBox runat="server"
        id="txtRange">
      </asp:TextBox>
      <br>
      <input type="submit" runat="server">
      <br>
      <asp:RangeValidator runat="server"
        ControlToValidate="txtRange"
        MaximumValue="10"
        MinimumValue="1">
```

Listing 11.3 continued

```

        ErrorMessage="Please enter a number between 1 and 10."
        Type="Integer">
    </asp:RangeValidator>
</form>
</body>
</html>

```

**Figure 11.3**

Validating a range using the RangeValidator control.

Advanced Validation Controls

In this section, you will explore the more advanced validation controls, such as the `RegularExpressionValidator` control, which is used to perform potentially complicated string-based input filtering. You also learn how to use the `CustomValidator` control to define your own validation conditions, and to use the `ValidationSummary` control to tie all page validation errors together in a consistent format.

The RegularExpressionValidator Control

The `RegularExpressionValidator` validates a user input based on a pattern defined by what is known as a *regular expression*. Regular expressions are extremely powerful, and have been around for a number of years in Perl and other programming languages. A complete discussion of regular expressions is beyond the scope of this book.

NOTE

An excellent resource for learning about the full breadth of the regular expression language can be found online at gnosis.cx/publish/programming/regular_expressions.html. Additionally, *Sams Teach Yourself Regular Expressions in 24 Hours*, by Jeffrey E. Friedl, is a fine resource.

A regular expression defines a pattern that the user input is expected to follow. It can be thought of as a kind of filter that the input must pass through in order for validation to succeed. For example, if you needed to allow a user to enter only vowels into a textbox, you would enter [aeiou] into the ValidationExpression property of the RegularExpressionValidator control.

Listing 11.4 demonstrates how to use the RegularExpressionValidator to make sure a user enters a U.S. Social Security number using the standard 000-00-0000 pattern.

Listing 11.4 RegularExpressionValidator.aspx—Advanced String-Based Validation with the RegularExpressionValidator Control

```
<html>
  <body>
    <form id="Form1" method="post" runat="server">
      Please enter your social security number:
      <br>
      <asp:TextBox runat="server"
        id="txtSocSecNumber">
      </asp:TextBox>
      <br>
      <input type="submit" runat="server">
      <br>
      <asp:RegularExpressionValidator runat="server"
        ControlToValidate="txtSocSecNumber"
        ValidationExpression=
          "[0-9][0-9][0-9]-[0-9][0-9]-[0-9][0-9][0-9]"
        ErrorMessage="Please use the following format: '000-00-0000' ">
    </asp:RegularExpressionValidator>
  </form>
</body>
</html>
```

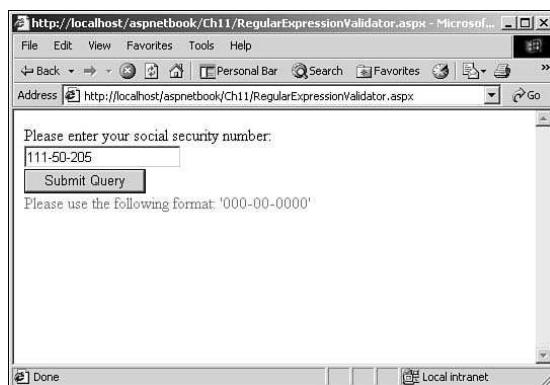


Figure 11.4

Using the RegularExpressionValidator control to validate a U.S. Social Security number.

The CustomValidator Control

When validation can't easily be performed using any of the other validation controls, the `CustomValidator` can be quite handy. The `CustomValidator` control enables you to specify your own functions that determine the validation criteria.

You use the `OnServerValidate` property of the `CustomValidator` control to specify the function that handles validating the control on the server side. Similarly, the `ClientValidationFunction` property specifies the function that handles validating the control on the client side. If possible, it is a good idea to use both; when no client-side validation function is available, the page will post to the server for validation. If no server-side validation function is specified, then the form data could pass validation and be posted to the server if the client has JavaScript disabled or is using a browser without JavaScript support.

Listing 11.5 demonstrates how to use the `CustomValidator` control to validate credit card numbers.

Listing 11.5 CreditCardValidator.aspx—This CustomValidator Control Is Used to Validate Credit Card Numbers

```
<html>
<head>
<script language="VB" runat="server">
Sub ServerValidate(s As Object, e As ServerValidateEventArgs)
    e.IsValid = ValidateCardVB(e.Value)
End Sub
Function ValidateCardVB(cc)
    ' Validates credit card number using
    ' check digit algorithm.
    ' Also checks length of number for Visa,
    ' MasterCard, and AmEx cards.
    ' Inputs: cc -- credit card to be validated.
    ' Outputs: returnvalue -- true if valid, false if invalid
    Dim intCheckSum, intLen, intProduct, intJ As Integer
    Dim strCh As String
    Dim fOK As Boolean

    ' Remove Extra Digits (spaces and dashes)
    intLen = Len(cc)
    For intJ = 1 To intLen
        strCh = Mid(cc, intJ, 1)
        If (strCh = " " Or strCh = "-") Then
            If intJ>1 Then
                cc = Left(cc,intJ-1) + Mid(cc,intJ+1)
            Else
                cc = Mid(cc,intJ+1)
            End If
        End If
    End If
End Function
```

Listing 11.5 continued

```
Next
intLen = Len(cc)
strCh = Left(cc,1)
If (strCh<>"3" And strCh<>"4" And strCh<>"5") Then
    'Must begin with 3, 4, or 5.
    fOK = False
ElseIf (strCh="5" And intLen<>16) Then
    'Mastercard must be 16 digits.
    fOK = False
ElseIf (strCh="4" And intLen<>13 And intLen<>16) Then
    'Visa numbers must be 13 or 16 digits.
    fOK = False
ElseIf (strCh="3" And intLen<>15) Then
    'American Express numbers must be 15 digits.
    fOK = False
Else
    ' Perform Checksum - Weighting list 2121212121212121....
    intCheckSum = 0
    For intJ = 1 To intLen ' go through entire cc num string
        ' convert char to int
        intProduct = CInt(Mid(cc,intJ,1))
        ' if odd digit from end, intProduct=intProduct*2
        If ((intLen-intJ) Mod 2 <> 0) Then
            intProduct=intProduct*2
        End If
        ' subtract 9 if intProduct is >=10
        If (intProduct >= 10) Then
            intProduct= intProduct - 9
        End If

        ' add to check
        intCheckSum= intCheckSum + intProduct
    Next

    ' card good if check divisible by 10
    If (intCheckSum Mod 10 = 0) Then
        'No error
        fOK = True
    Else
        'Invalid check sum
        fOK = False
    End If
End If

Return fOK
End Function
```

</script>

Listing 11.5 continued

```
<script language="javascript">
function ClientValidate(s, e) {
    e.IsValid = ValidateCardJS(e.Value)
}
function ValidateCardJS(cc) {
    /* Validates credit card number using
    check digit algorithm.
    Also checks length of number for Visa,
    MasterCard, and AmEx cards.
    Inputs: cc -- credit card to be validated.
    Outputs: returnvalue -- true if valid, false if invalid */
    var intCheckSum
    var intLen
    var intProduct
    var intJ
    var strCh
    var fOK

    // Remove Extra Digits (spaces and dashes)
    intLen = cc.length
    for(intJ=0; intJ<intLen; intJ++) {
        strCh = cc.charAt(intJ)
        if (strCh == " " || strCh == "-") {
            if(intJ>0)
                cc = cc.substring(0,intJ) + cc.substring(intJ+1, cc.length)
            else
                cc = cc.substring(intJ+1, cc.length)
        }
    }
    // Recomputed length after stripping spaces & dashes
    intLen = cc.length

    // Check first character
    strCh = cc.charAt(0)
    if (strCh!="3" && strCh!="4" && strCh!="5") {
        //Must begin with 3, 4, or 5
        fOK = false
    }
    else if (strCh=="5" && intLen!=16) {
        //Mastercard must be 16 digits.
        fOK = false
    }
    else if (strCh=="4" && intLen!=13 && intLen!=16) {
        //Visa numbers must be 13 or 16 digits.
        fOK = false
    }
    else if (strCh=="3" && intLen!=15) {
```

Listing 11.5 continued

```
//American Express numbers must be 15 digits.  
fOK = false  
}  
else {  
    // Perform Checksum - Weighting list 2121212121212121....  
    intCheckSum = 0  
  
    // go through entire cc num string  
    for (intJ=0;intJ<intLen;intJ++) {  
        // convert char to int  
        intProduct = parseInt(cc.charAt(intJ))  
        // if odd digit from end, intProduct=intProduct*2  
        if ((intLen-intJ-1) % 2 != 0)  
            intProduct *= 2  
        // subtract 9 if intProduct is >=10  
        if (intProduct >= 10)  
            intProduct -= 9  
  
        // add to check  
        intCheckSum += intProduct  
    }  
    // card good if check divisible by 10  
    if (intCheckSum % 10 == 0) {  
        // No error  
        fOK = true  
    }  
    else {  
        // Invalid check sum  
        fOK = false  
    }  
}  
return fOK  
}  
  
</script>  
  
</head>  
<body>  
  
<form runat="server">  
    <table>  
        <tr>  
            <td>Name (as it appears on card):</td>  
            <td><asp:TextBox id="txtCardName"  
                    columns=30 runat="server" /></td>  
        </tr>
```

Listing 11.5 continued

```

<tr>
<td>Credit Card Number:</td>
<td><asp:TextBox id="txtCreditCard"
    columns=30 runat="server" /></td>
</tr>

<tr>
<td>Expiration Date (Month/Year):</td>
<td><asp:TextBox id="txtCardMo"
    columns=4 runat="server" /> /
<asp:TextBox id="txtCardYr"
    columns=4 runat="server" /> </td>
</tr>

<tr>
<td><asp:Button type="submit" text="Submit" runat="server" /></td>
</tr>
</table>
<asp:CustomValidator id="cvCreditCard" runat="server"
    ControlToValidate="txtCreditCard"
    OnServerValidate="ServerValidate"
    ClientValidationFunction="ClientValidate"
    Display="Static"
    ErrorMessage="Invalid credit card number."
    EnableClientScript=True
    Font-Name="verdana" Font-Size="14pt">
</asp:CustomValidator>
</form>
</body>
</html>

```

The example uses both client-side and server-side validation functions that ensure the user enters a valid Visa, MasterCard, or American Express credit card number.

If ASP.NET determines that client-side JavaScript is supported by the browser, the `cvCreditCard` `CustomValidator` control calls the `ClientValidate` function, which in turn calls the `ValidateCardJS` JavaScript credit card validation function. In cases where the client-side code cannot be used, the `cvCreditCard` `CustomValidator` control calls the `ServerValidate` function, which in turn calls the `ValidateCardVB` function.

The `ValidateCardJS` and `ValidateCardVB` functions perform the same set of checks on the credit card number. First they strip out any spaces or hyphens and then they check to see that the credit card number begins with 3, 4, or 5. The first digit of the card number indicates the type of card. Next, the routines check the length of the credit card number to verify that the number is the correct length. Finally, a checksum test is performed on the credit card number using a weighting factor. If the resulting checksum is evenly divisible by 10 the credit card number is determined to be valid.

Both the client and server functions are passed two arguments: source and arguments. The second parameter, arguments, has two properties: Value and IsValid. You use the Value property to grab the value of the ControlToValidate control. You use IsValid to let the CustomValidator control know whether the custom validation code determined if the value was valid.

For example, the ClientValidate function from CreditCardValidator.aspx sets the value of the IsValid property to the return value of the ValidateCardJS function for the client-side validation scenario:

```
function ClientValidate(s, e) {  
    e.IsValid = ValidateCardJS(e.Value)  
}
```

Here's the analogous code from CreditCardValidator.aspx that sets the value of the IsValid property to the return value of the ValidateCardVB function for the server-side validation scenario:

```
Sub ServerValidate(s As Object, e As ServerValidateEventArgs)  
    e.IsValid = ValidateCardVB(e.Value)  
End Sub
```

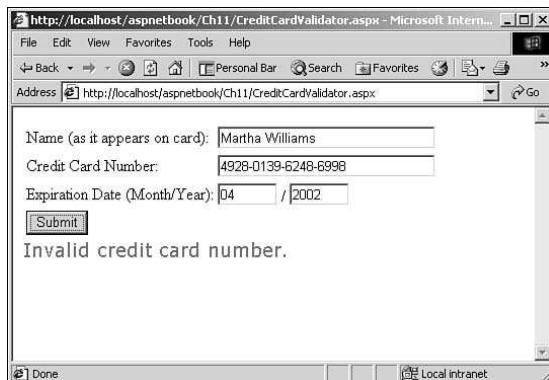


Figure 11.5

This page employs the CustomValidator control and both client and server-side custom code to validate credit card numbers.

NOTE

The ValidateCardJS and ValidateCardVB functions will validate Visa, MasterCard, and American Express credit card numbers to determine that they meet length and checksum requirements. They do not, of course, verify that the referenced credit card number is real. That would require the services of a credit card service.

The ValidationSummary Control

The ValidationSummary control is used to group error messages from several validation controls in a single place. This gives you a great deal of flexibility. In fact, you can completely hide all validation controls on a page (by setting each control's Display property to None) and then use the ValidationSummary control to display the errors.

The ValidationSummary control is interesting in that it doesn't actually perform any validation on its own. It's not derived from the BaseValidator class like the rest of the controls covered in this chapter and therefore technically isn't a validation control.

The ValidationSummary control has a few unique properties that control the format and type of display for the error messages. The DisplayMode property enables you to display the validation errors in either a BulletList, standard List, or in a SingleParagraph format. Additionally, if you set the ShowMessageBox property to True, then a message box containing the page validation errors is displayed to the user when the user clicks the submit button.

Listing 11.6 contains an example that shows a simple login page. The ValidationSummary control is used to display a bulleted list of all validation errors that have occurred on the page. Additionally, a message box is displayed to the user if there are any errors.

Listing 11.6 ValidationSummary.aspx—Bringing It All Together with the ValidationSummary Control

```
<html>
<body>
<form id="Form1" method="post" runat="server">

<asp:ValidationSummary runat="server"
    DisplayMode="BulletList"
    ShowMessageBox="True">
</asp:ValidationSummary>
<table>
<tr>
    <td>Login ID:</td>
    <td><asp:TextBox id="txtLogin" runat="server" />
<asp:RequiredFieldValidator Runat="server"
    ControlToValidate="txtLogin"
    ErrorMessage="Please enter your Login ID">*</asp:RequiredFieldValidator></td>
</tr>

<tr>
    <td>Password:</td>
    <td><asp:TextBox id="txtPassword"
        TextMode="Password" runat="server" />
<asp:RequiredFieldValidator Runat="server"
    ControlToValidate="txtPassword"
    ErrorMessage="Please enter your Password">*</asp:RequiredFieldValidator></td>
</tr>
```

Listing 11.6 continued

```
ControlToValidate="txtPassword"
ErrorMessage="Blank passwords are not permitted">*
</asp:RequiredFieldValidator></td>
</tr>

<tr>
<td><input type="submit" runat="server" /></td>
</tr>
</table>
</form>
</body>
</html>
```

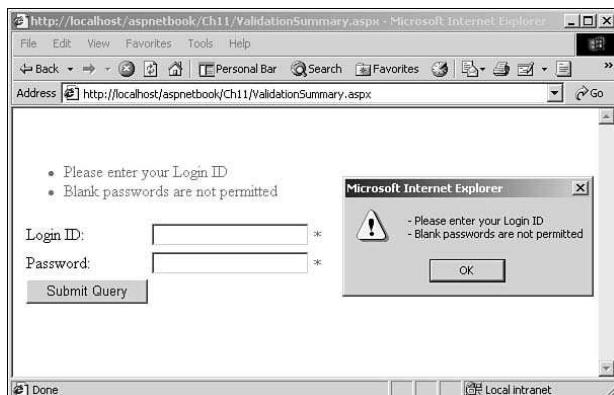


Figure 11.6

You can use the ValidationSummary control to aggregate the error messages from multiple validation controls.

Summary

In this chapter, you've seen how to use validation controls to build more robust Web forms. You explored the six validation controls: RequiredFieldValidator, CompareValidator, RangeValidator, RegularExpressionValidator, CustomValidator, and ValidationSummary. In just a few lines of code, validation controls enable you to easily verify user input and prevent errors due to unpredictable user input.

CHAPTER 12

Adding User Controls to Your Web Forms

Throughout the last several chapters, you learned how to take advantage of the new Web controls that ship as part of the ASP.NET framework classes. These Web form controls are built on a powerful object model that allows you to create user interfaces that are easy to program and maintain. In this chapter, you'll learn how to use that same object model to create your own specialized controls and incorporate them into your ASP.NET solutions.

What Are User Controls?

User controls are an easy way to create reusable visual components for your ASP.NET solutions. Just as you create ASP.NET pages using a mix of markup code and execution code in C#, Visual Basic, or other languages, you use the same techniques to build user controls. However, while ASP.NET pages are saved using the `.ASPx` file extension, user controls are saved using the `.ASCX` file extension. Also, the register directive at the top of User controls starts with the `control` keyword instead of the `page` keyword. Aside from these minor differences, creating ASP.NET pages and user controls works about the same.

One of the primary advantages of creating user controls is that it allows you to re-use functionality across multiple pages in your ASP.NET solution. In classic ASP, you had the ability to use separate include files to do almost the same thing. However, the classic ASP include files were always “global” in scope and could slow execution of the interpreted pages. Finally, include files were not able to handle their own events. This meant that using include files required quite a bit of coordination between the host page and the include file itself.

NOTE

Include files are still possible in ASP.NET and work pretty much the same way as they do in classic ASP. However, in many cases, User Controls provide the same functionality along with additional protection and possibilities.

User controls, on the other hand, are self-contained components. This means that they can handle their own events, publish custom properties, and have a limited execution scope. It is now much easier to create and use user controls to do the tasks typically handled by include files in the past.

Simple User Controls

Creating user controls is as easy as creating a document with basic HTML markup and incorporating it into your ASPX page. You can also add custom attributes to the controls to allow you to safely and easily pass parameters into the user control for processing.

In this section, you'll learn how to create and use this simple form of user controls.

Creating a Markup-Only User Control

The most basic form of a user control is one that contains just simple HTML markup. This allows you to quickly and easily create re-usable components for your ASP.NET solutions.

There are two basic steps to creating user controls. First, you need to create a new file in your project that ends with the .ASCX extension. This is the only valid extension that you can use to create user controls for ASP.NET. For example, mycontrol.ascx would be a valid name for a user control.

Listing 12.1 shows a simple user control that contains only HTML markup. This control contains a simple menu of external links.

Listing 12.1 MARKUP.ASCX—A Simple User Control That Contains Only HTML Markup

```
<%@ control classname="MarkUpControl" %>

<font color="Green">
<h3>Site Links</h3>
</font>
<ul>
    <li><a href="http://www.microsoft.com">Microsoft</a></li>
    <li><a href="http://msdn.microsoft.com">MSDN</a></li>
    <li><a href="http://www.gotdotnet.com">GotDotNet</a></li>
    <li><a href="http://www.asp.net">ASP.NET</a></li>
    <li><a href="http://www.eraserver.net">EraServer.NET</a></li>
</ul>
```

Notice the control directive at the top of the page. This contains the `classname` attribute. This attribute is optional, but it's a good idea to add it to all your user controls.

This identifies the class name that ASP.NET should use when dynamically compiling your control. You can also use this class name for advanced tasks such as loading the control dynamically at runtime.

Once the user control is created and saved to a Web, you'll be able to incorporate that user control into an existing ASPX document. To do this, you need to add a `register` directive to the top of every ASPX page that will access the user control. This directive will instruct the ASP.NET engine to load and parse your user control at compile time. The following is a typical `register` directive that references a user control.

```
<%@ register tagprefix="aspx4devs" tagname="MarkUp" src="MarkUp.ascx" %>
```

In the line just shown, there are three attributes that are important. First, the `tagprefix` attribute is used to identify the namespace prefix used in the markup of the page. This tells ASP.NET which controls on the following page actually refer to the `markup.ascx` control. Next, the `tagname` attribute is used to give the user control an object name on the page. You use this object name when you write any code to manipulate the user control. Finally, the `src` attribute tells the ASP.NET runtime where to find the source code file that contains the user control.

Once the `register` tag is added, you can place the actual markup that references the user control in your page. Listing 12.2 shows how this would look for the `markup.ascx` user control shown earlier.

Listing 12.2 MARKUPHOST.ASPX—Hosting the MarkUp User Control

```
<%@ Page Description="hosting a user control" %>
<%@ register tagprefix="aspx4devs" tagname="MarkUp" src="MarkUp.ascx" %>

<html>
  <body>
    <h2>User Control Examples</h2>
    <hr />

    <form runat="server">
      <table width="100%">
        <tr>
          <td width="20%" valign="top">
            <aspx4devs:MarkUp runat="server" />
          </td>
          <td width="80%" valign="top">
            <h3>Main Content Here</h3>
          </td>
        </tr>
      </table>
    </form>

  </body>
</html>
```

You can see that both the register directive and the actual user control markup must appear on the page in order for the control to appear at runtime (see Figure 12.1).

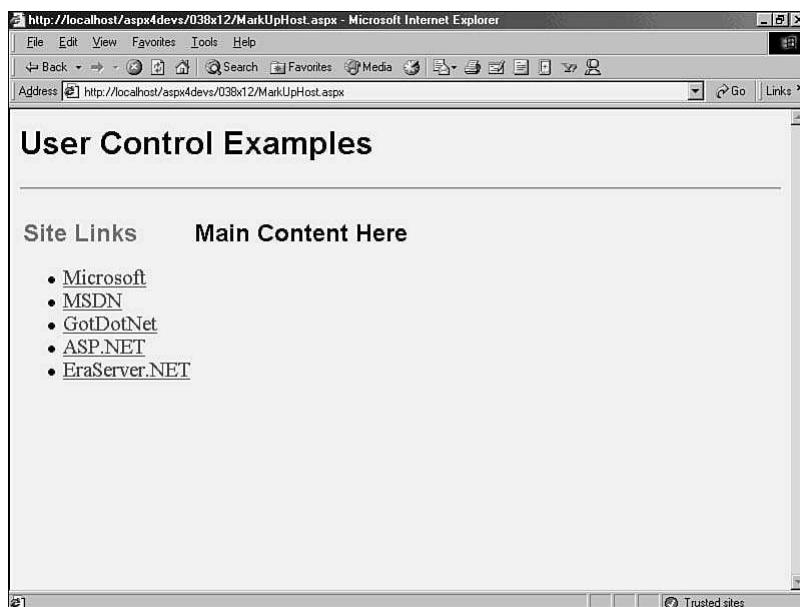


Figure 12.1

Testing the *MarkUp* user control.

Adding Custom Properties to User Controls

You can also add custom properties to user controls. Custom properties are a great way to safely pass parameters into user controls at runtime. You can set the properties directly on the user control in the host page using attributes. You can also set the properties within the server-side script block of the hosting page.

The first step in creating custom properties for a user control is adding the public properties to the user control itself. Listing 12.3 shows an example of a user control with custom properties.

Listing 12.3 PROPERTIES.ASCX—Adding Custom Properties to a User Control

```
<%@ control classname="PropertiesControl" %>

<script language="vb" runat="server">

public TitleText as string="Site Links"
public TitleColor as string="green"

sub Page_Load(sender as object, args as EventArgs)
```

Listing 12.3 continued

```

thisTitle.Text=TitleText
thisFont.Attributes("color")=TitleColor

end sub

</script>

<font id="thisFont" runat="server">
<h3><asp:Label id="thisTitle" runat="server"/></h3>
</font>
<ul>
<li><a href="http://www.microsoft.com">Microsoft</a></li>
<li><a href="http://msdn.microsoft.com">MSDN</a></li>
<li><a href="http://www.gotdotnet.com">GotDotNet</a></li>
<li><a href="http://www.asp.net">ASP.NET</a></li>
<li><a href="http://www.eraserver.net">EraServer.NET</a></li>
</ul>

```

As you can see from Listing 12.3, all you need to do is create public variables within a script block of the user control file. These public variables are then treated as custom properties for the control.

You can also see that the user control has its own `Page_Load` event. It is in this event that you can inspect the contents of the custom properties and use their values to alter the appearance of the control. In the case of the control in Listing 12.3, the custom properties are used to set the title and color of the menu header text.

Listing 12.4 shows how an ASPX page would host the control defined in Listing 12.3 and how you can set the custom properties of the user control.

Listing 12.4 PROPERTIESHOST.ASPX—Setting the Custom Properties of a User Control

```

<%@ Page Description="hosting a user control" %>
<%@ register tagprefix="aspx4devs"
   tagname="Properties"
   src="Properties.ascx" %>

<html>
<body>

<h2>User Control Examples</h2>
<hr />

<form runat="server">
  <table width="100%">
    <tr>
      <td width="20%" valign="top">

```

Listing 12.4 continued

```
<asp:Properties>
    <TitleText>Properties</TitleText>
    <TitleColor>red</TitleColor>
</asp:Properties>
</td>
<td width="80%" valign="top">
    <h3>Main Content Here</h3>
</td>
</tr>
</table>
</form>

</body>

</html>
```

The resulting output from the page in Listing 12.3 is shown in Figure 12.2.

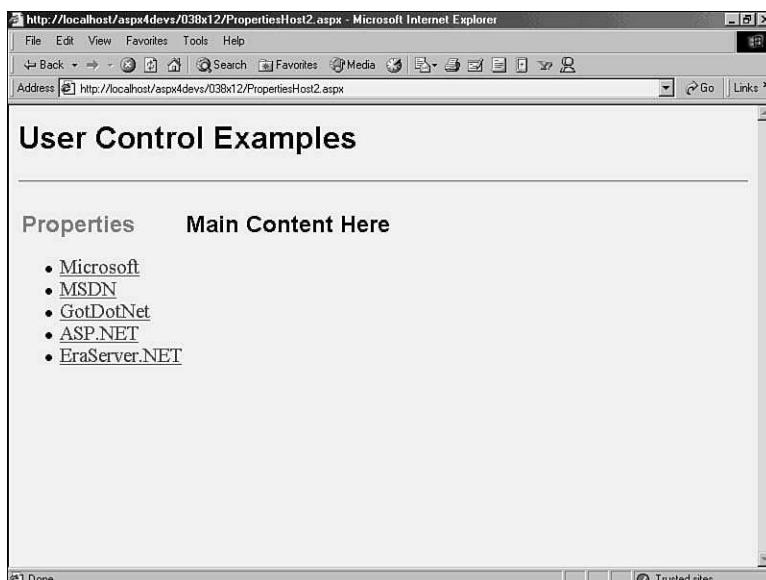


Figure 12.2

Results of showing the Properties user control.

It is important to point out that you can also use server-side script to set the properties of a custom control. For example, the following code shows how you could set the `TitleText` and `TitleColor` properties of the `Properties` user control named "myProp" in a server-side script block.

```
<script Language="VB" runat="server">
    sub Page_Load(sender as object, e as EventArgs)
        myProp.TitleText = "Properties"
        myProp.TitleColor = "red"
    end sub
</script>
```

```
myProp.TitleText="Properties"  
myProp.TitleColor="red"  
  
end sub  
  
</script>
```

Now that you understand the basics of user controls, you can move on to more advanced topics such as event handling and loading user controls dynamically at runtime.

Advanced Features of User Controls

Along with the ability to encapsulate markup and properties, user controls also allow you to handle your own events and even dynamically load and parse the controls at runtime.

This section shows you how you can take advantage of these features in your own user controls.

Handling Events in User Controls

One really powerful aspect of user controls is the ability to handle your own events within the control. For example, you can add a search dialog to a user control and then place the code for the click event right in the user control. This reduces the amount of code in the main page and places the event-handler much closer to the markup to which it belongs.

Listing 12.5 shows a search dialog that has been added to a user control. You can also see that the click event handler is also contained in the user control file.

Listing 12.5 EVENTS.ASCX—Handling Events Within a User Control

```
<%@ control classname="EventsControl" %>  
  
<script language="vb" runat="server">  
  
public TitleText as string="Site Links"  
public TitleColor as string="green"  
  
sub Page_Load(sender as object, args as EventArgs)  
  
    thisTitle.Text=TitleText  
    thisFont.Attributes("color")=TitleColor  
  
end sub  
  
sub search_click(sender as object, args as EventArgs)  
  
    searchResults.Text="Item not found!"  
  
end sub  
  
</script>
```

Listing 12.5 continued

```
<font id="thisFont" runat="server">
<h3><asp:Label id="thisTitle" runat="server" /></h3>
</font>
<ul>
  <li><a href="http://www.microsoft.com">Microsoft</a></li>
  <li><a href="http://msdn.microsoft.com">MSDN</a></li>
  <li><a href="http://www.gotdotnet.com">GotDotNet</a></li>
  <li><a href="http://www.asp.net">ASP.NET</a></li>
  <li><a href="http://www.eraserver.net">EraServer.NET</a></li>
</ul>

<p align="center">
<b>Search</b><br />
<asp:TextBox id="searchInput" size="15" runat="server" /><br />
<asp:Button id="searchButton" Text="Submit"
  OnClick="search_click" runat="server" /><br />
<font color="red"><asp:Label id="searchResults" runat="server" /></font>
</p>
```

It is important to point out that, while local event handling is a feature of the user controls, it is *not* an optional one. In other words, any user controls that have an event handler call (as does the control in Listing 12.5), must also have the execution code defined as well. User controls were not designed to allow programmers to pass event messages from the user control up to the parent control.

NOTE

If you want to create controls that can pass event messages to the parent control, you can create Custom Controls using Visual Basic .NET. Custom controls are not covered in this book, but are covered in the online documentation that ships with ASP.NET.

Once you have defined the user control that handles its own events, you can host it inside an ASPX page. Listing 12.6 shows an example page that hosts the EVENTS.ASCX control shown in Listing 12.5.

Listing 12.6 EVENTSHOST.ASPX—Hosting a User Control That Has Its Own Event Handler

```
<%@ Page Description="hosting a user control" %>
<%@ register tagprefix="aspx4devs" tagname="Events" src="Events.ascx" %>

<html>
<body>

<h2>User Control Examples</h2>
<hr />

<form runat="server">
```

Listing 12.6 continued

```

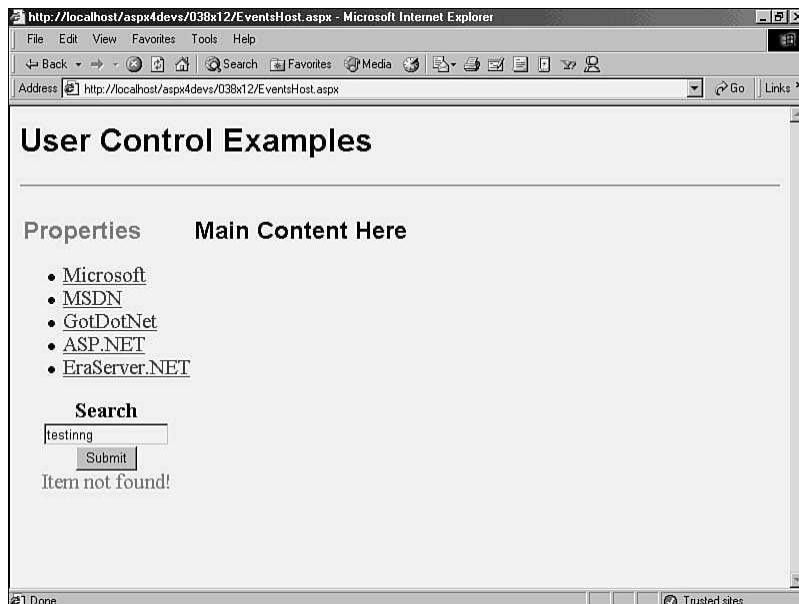
<table width="100%">
    <tr>
        <td width="20%" valign="top">
            <asp:Events
                TitleText="Properties"
                TitleColor="red" runat="server" />
        </td>
        <td width="80%" valign="top">
            <h3>Main Content Here</h3>
        </td>
    </tr>
</table>
</form>

</body>
</html>

```

You should notice that the code from Listing 12.6 looks no different from the code in Listing 12.4. That is because the existence of the search dialog in the EVENTS.ASCX control is of no real importance to the hosting page. The details of the search dialog are all handled by the user control itself.

Figure 12.3 shows how this looks in a browser.

**Figure 12.3**

Testing the Events user control.

Loading User Controls Dynamically

The last feature of user controls that will be covered in this chapter is the ability to load them dynamically at runtime. This allows you to easily create data-driven pages that can be composed of any number of controls based on data inputs and other factors known only after the program has already been compiled and is running.

In the example shown in Listing 12.7, a user control allows programmers to pass LinkURL and LinkText properties in order to create a menu entry for a list. This single control can be loaded any number of times depending on the number of links to display on a given page.

Listing 12.7 DYNAMIC.ASCX—Creating a User Control for Dynamic Loading

```
<%@ control classname="DynamicControl" %>

<script language="vb" runat="server">

public LinkText as string="Some Link"
public LinkURL as string="#"

sub Page_Load(sender as object, args as EventArgs)

    thisPlace.InnerHtml = "<a href=''" & LinkURL & _
        "'><b>" & LinkText & "</b></a><br />"

end sub

</script>

<span id="thisPlace" runat="server"/>
```

Now that you have a control that performs a simple task (adding links to a page), you can create an ASPX page that dynamically loads copies of this control at runtime. Listing 12.8 shows how this might look in an ASPX page.

Listing 12.8 DYNAMICHOST.ASPX—Loading User Controls Dynamically

```
<%@ Page Description="hosting a user control" %>
<%@ register tagprefix="aspx4devs" tagname="Dynamic" src="Dynamic.ascx" %>

<script Language="VB" runat="server">

sub Page_Load(sender as object, e as EventArgs)

    dim x as integer = 0
    dim aList() as string = _
        {"http://www.microsoft.com", "Microsoft", _
```

Listing 12.8 continued

```

"http://www.asp.net", "ASP.NET", _
"http://www.gotdotnet.com", "GotDotNet", _
"http://www.EraServer.NET", "EraServer.NET" }

for x = 0 to aList.Length-1 step 2
    dim c as Control = LoadControl("dynamic.ascx")
    CType(c, DynamicControl).LinkURL=aList(x)
    CType(c, DynamicControl).LinkText=aList(x+1)
    thisPlace.Controls.Add(c)
next

end sub

</script>

<html>
<body>

<h2>User Control Examples</h2>
<hr />

<form runat="server">
    <table width="100%">
        <tr>
            <td width="20%" valign="top">
                <h3>Site Links</h3>
                <asp:PlaceHolder id="thisPlace" runat="server"/>
            </td>
            <td width="80%" valign="top">
                <h3>Main Content Here</h3>
            </td>
        </tr>
    </table>
</form>

</body>
</html>

```

Listing 12.8 shows how you can use the `LoadControl` method to dynamically load an instance of the user control into the page. You can also see the use of the `<asp:PlaceHolder />` control. This placeholder control acts as the “container” for the added user controls. This is an important point. When you dynamically add controls, to a page, you always need to have an existing target control to “hold” the new controls. That is what the placeholder is used for—a target for loading controls on a page.

Figure 12.4 shows the results of running the `DynamicHost.aspx` page in a browser.

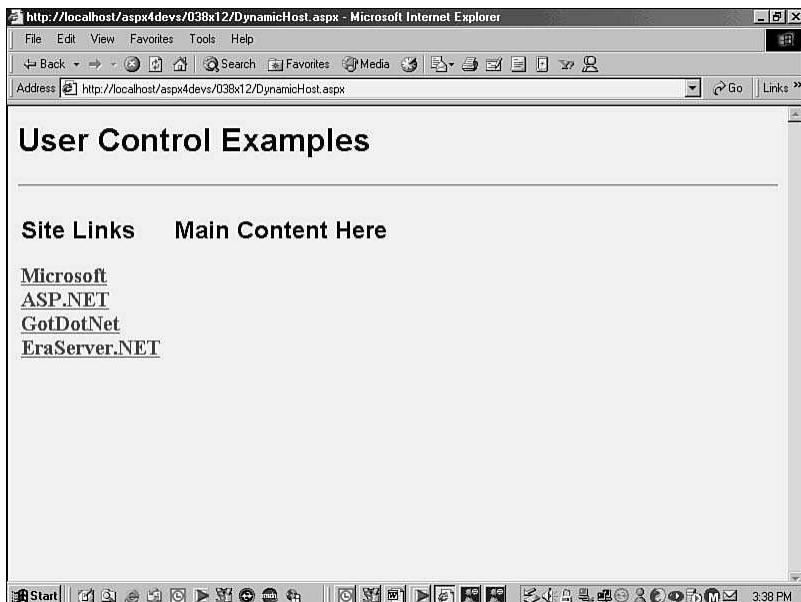


Figure 12.4

Testing the Dynamic.ASCX user control.

Summary

In this chapter you learned how you can create your own user controls for ASP.NET that allow you to encapsulate HTML markup, pass parameters, handle your own events, and even how to load these control dynamically at runtime. This gives you the ability to create self-contained re-usable components for your ASP.NET solutions.

PART IV

HANDLING DATA ACCESS WITH ADO.NET

- 13 Introduction to ADO.NET and Data Binding
- 14 Accessing Data with .NET Data Providers
- 15 Working with ADO.NET DataSets

CHAPTER 13

Introduction to ADO.NET and Data Binding

Data access in ASP.NET revolves around a set of .NET Framework classes referred to as *ADO.NET*. This set of classes represents the next version of ActiveX Data Objects (ADO). While maintaining some consistency with the existing ADO object model—the Connection and Command classes, for example, work similarly across both object models—ADO.NET differs significantly from ADO “classic” in a number of ways, including

- ADO “classic” works in a connected scenario. ADO.NET has been architected to work in a disconnected scenario.
- ADO.NET separates the data container classes from database access classes. In ADO “classic,” these two sets of classes are intermingled.
- ADO “classic” sits on top of OLE DB. All data access works against OLE DB providers. ADO.NET works with both OLE DB and non-OLE DB providers, and ships with the SQL Server .NET Data Provider, which does not involve OLE DB.
- The ADO.NET `DataSet` class can hold multiple tables of data in `DataTable` objects. It even supports relationships and views based on the `DataTables`. In contrast, the ADO “classic” `Recordset` is architected to hold only one result set; there is a way to create multiple unrelated result sets, but it’s much less useful.
- When updating data in ADO “classic” `Recordsets`, you have no control over how the updates are executed. In ADO.NET, when updating data stored in `DataSet`s, you can specify how the updates are executed and, in fact, direct ADO.NET to employ custom stored procedures to perform the updates.

- ADO.NET directly supports the execution of queries returning a single value without the need to create a `DataSet`. ADO “classic” has no such support.
- ADO.NET has rich support for moving data between its `DataSets` and XML files and schemas. ADO “classic” has very limited support for XML.

The ADO.NET Dichotomy

ADO.NET is made up of five namespaces, summarized in Table 13.1.

Table 13.1 The ADO.NET Namespaces

Class	Purpose
<code>System.Data</code>	Contains classes that hold data, disconnected from any data source. Key classes include <code>DataSet</code> , <code>DataTable</code> , <code>DataRow</code> , and <code>DataView</code> .
<code>System.Data.Common</code>	Contains common base classes that are inherited by the OLE DB and SQL Server .NET Data Providers. You won’t work directly with these classes.
<code>System.Data.OleDb</code>	Contains the database classes for working with OLE DB data providers. Key classes include <code>OleDbConnection</code> , <code>OleDbCommand</code> , <code>OleDbDataReader</code> , and <code>OleDbDataAdapter</code> .
<code>System.Data.SqlClient</code>	Contains the database classes for working with SQL Server via its native Tabular Data Stream (TDS) protocol. Key classes include <code>SqlConnection</code> , <code>SqlCommand</code> , <code>SqlDataReader</code> , and <code>SqlDataAdapter</code> .
<code>System.Data.SqlTypes</code>	Contains classes for working with SQL Server native data types.

The core namespaces of ADO.NET are `System.Data`, `System.Data.OleDb`, and `System.Data.SqlClient`. In general, you won’t work directly with the `System.Data.Common`. This namespace is provided primarily for vendors creating new data providers. In addition, the `System.Data.SqlTypes` namespace plays a fairly minor role in mapping SQL Server data types to .NET data types.

A dichotomy in ADO.NET arises from the fact that the `System.Data` namespace and its primary class, the `DataSet` class, is concerned only with in-memory storage of data and knows nothing about data access or databases. In contrast, the `System.Data.OleDb` and `System.Data.SqlClient` namespaces concern themselves primarily with data access and connecting to databases to read and write records.

The ADO.NET dichotomy is depicted schematically in Figure 13.1. On the left are the classes from the .NET data provider namespaces (`System.Data.OleDb` or `System.Data.SqlClient`), which are concerned with communicating with relational databases (as well as other sources of data). On the right are the classes provided by the `System.Data` classes, which are concerned with in-memory data storage and reading and writing of XML.

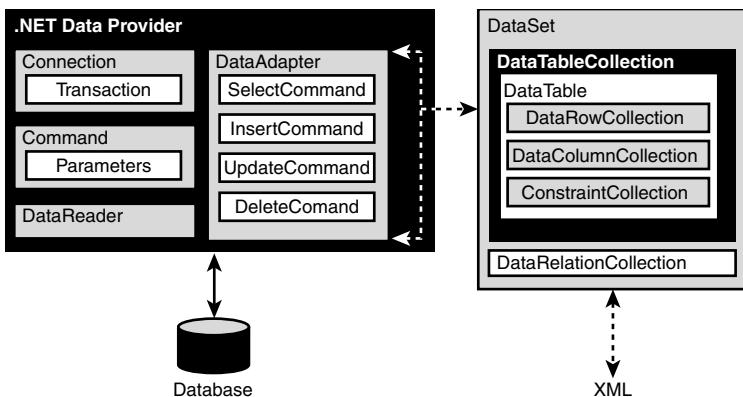


Figure 13.1

The ADO.NET object model.

The Database Classes: Data Providers

The classes of the `System.Data.SqlClient` and `System.Data.OleDb` namespaces are responsible for accessing data from SQL Server and OLE DB data sources, respectively. These classes include the `Connection`, `Command`, `DataReader`, and `DataAdapter` classes. These are the generic class names. When working with these classes, you will either prefix each class with “`Sq1`,” to use the SQL Server .NET data provider classes, or “`OleDb`,” to use the OLE DB data provider classes.

- You use the `Connection` classes to establish a connection to the appropriate data provider.
- You use the `Command` classes to execute SQL statements and stored procedures.
- If a `Command` class executes a SQL statement or stored procedure returns records, you can use the `DataReader` classes to provide a read-only, forward-only cursor through the records.
- You use the `DataAdapter` classes to execute an SQL statement or stored procedure and fill a `DataSet` (which is part of the `System.Data` namespace) with records. Going the other way, you can also use the `DataAdapter` classes to scan a `DataSet` for updated rows and apply those changes to the database.

The database classes are discussed in much greater detail in Chapter 14, “Accessing Data with .NET Data Providers.”

The Data Classes: The DataSet

The classes of the `System.Data` namespace are responsible for in-memory, disconnected data storage. The primary class of this namespace is the `DataSet` class. The `DataSet` class is the container for a number of other classes, including the `DataTable` class which contains rectangular sets of data (also known as tables). `DataTables` contain `DataRow`s, individual rows of data. `DataSet`s may also contain `DataView`s, filtered and sorted views of `DataTables`, and `DataRelation`s, relationships between the `DataTables`.

You can create a `DataSet` and fill it in a variety of ways. You can construct the `DataSet` and its data entirely from code. You can also create the `DataSet` from an XML file. More typically, however, you'll create a `DataSet` from database data using one of the `DataAdapter` classes. `DataSet`s are not just one-way containers, however, you can update their contents and then pass the updates back to a database (again using the `DataAdapter` classes) or to an XML file.

The classes of the `System.Data` namespace are the subject of Chapter 15, “Working with ADO.NET `DataSet`s.”

Data Binding

ASP.NET provides very flexible and robust binding support of data to just about any control you can place on a page. In fact, you don't even need a control—you can even bind to HTML bits on a page.

To bind to a data source you need to do two things:

1. Use the ASP.NET data binding syntax to specify a property or literal text on the page that you wish to bind to a data binding expression. The data binding syntax is

```
<%# bindingexpression %>
```

2. Use the `.DataBind` method of the `Page` object or a control to bind the data source to the data binding expression.

Data binding does not require an ADO.NET data source. You can bind to a property, method, function, or collection.

Binding to Properties, Methods, and Functions

The code shown in Listing 13.1, taken from `.DataBindSimple.aspx`, illustrates binding to properties and methods.

Listing 13.1 DataBindSimple.aspx—With ASP.NET Data Binding, You Can Bind to Just About Anything, Including Properties and Methods

```
<html>
<head>
<title>.DataBindSimple.aspx</title>
<style>
    body {font-size: 14pt;}
    input {font-size: 12pt;}
</style>
<script language="VB" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
    Page.DataBind
End Sub
```

Listing 13.1 continued

```

Class Stuff
ReadOnly Shared Property RandomNumber() As String
    Get
        Dim rnd As System.Random = New System.Random
        Return rnd.Next(1,100)
    End Get
End Property

Public Shared Function GetNow() As DateTime
    Return DateTime.Now
End Function
End Class
</script>
</head>

<body>
<form runat="server">
<p><b>Time:</b> <asp:Textbox id="txtTime" runat="server"
   Text="<%# Stuff.GetNow() %>" /></p>
</form>
<p><b>Random Number:</b> <%# Stuff.RandomNumber %></p>
</body>
</html>

```

The DataBindSimple.aspx example from Listing 13.1 binds the `Text` property of a `Textbox` server control to the `GetNow` property of the `Stuff` class:

```
<asp:Textbox id="txtTime" runat="server"
Text="<%# Stuff.GetNow() %>" />
```

For the binding to occur at runtime, you must invoke the `Page` class's `DataBind` method, as shown here:

```

Sub Page_Load(sender As Object, e As EventArgs)
    Page.DataBind
End Sub

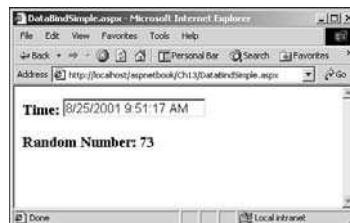
```

Binding to a method is just as simple. Also found in Listing 13.1, the following code sets up the binding of literal HTML to the `RandomNumber` method of the `Stuff` class:

```
Random Number:</b> <%# Stuff.RandomNumber %>
```

Binding to a function that is not a member of a class module is analogous to binding to a class method.

`.DataBindSimple.aspx` is shown in Figure 13.2.

**Figure 13.2**

This page demonstrates how to bind to properties and methods.

Binding to Collections

ASP.NET supports data binding to arrays and most collection classes. In fact, you can bind to any collection or list class that supports the `IEnumerable`, `ICollection`, or `IListSource` interface. Classes that support these interfaces include the `Array`, `ArrayList`, `Hashtable`, `DataView`, and `DataReader`.

The code in Listing 13.2, from `DataBindArray.aspx`, illustrates how to data bind to an array.

Listing 13.2 DataBindArray.aspx—This Page Illustrates How to Bind a DropDownList Server Control to an Array

```
<script language="VB" runat="server">
Sub Page_Load(sender As Object, e As EventArgs)
    If Not IsPostBack Then
        Dim aColors(4) as String

        aColors(0) = "Red"
        aColors(1) = "Green"
        aColors(2) = "Blue"
        aColors(3) = "Black"
        aColors(4) = "Yellow"

        drpColors.DataSource = aColors
        drpColors.DataBind
    End If
End Sub

Sub cmdSelect_Click(sender As Object, e As EventArgs)
    Dim strColor As String = drpColors.SelectedItem.Text
    lblOut.Text = "You chose: " & _
        "<font color=" & strColor & ">" & strColor & "</font>"
End Sub

</script>
</head>
```

Listing 13.2 continued

```
<body>
<form runat="server">
    <asp:Label id="lblTitle" runat="server"
        Text="<h2>Array Data Binding Example</h2>" />
    <p><asp:DropDownList id="drpColors" runat="server" />
    <asp:button id="cmdSelect" runat="server"
        Text="Select" OnClick="cmdSelect_Click" /></p>
    <p><asp:Label id="lblOut" runat="server" /></p>
</form>
```

The page produced by DataBindArray.aspx is shown in Figure 13.3.

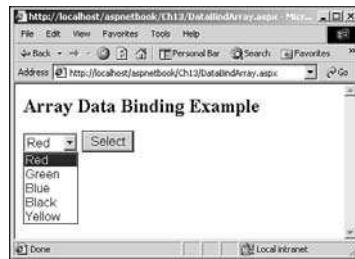


Figure 13.3

This DropDownList server control is bound to an array.

Binding Complex List Controls to the DataSet Class

The ADO.NET `DataSet` class is a convenient class to which to bind. The `DataSet` class provides an in-memory container for data. `DataSets` themselves do not directly contain rows of data. Instead they are containers for `DataTable`s which actually contain the rows and columns of data. `DataSets` also contain `DataView` objects which you can use to filter and sort the data found in `DataTable`s. In addition, every `DataTable` object has a `DefaultView` property that returns—you guessed it—the default view of the `DataTable`.

Neither the `DataSet` nor the `DataTable` class support direct data binding. The `DataView` object, however, implements the necessary interfaces to support data binding.

Binding to a `DataGridView` Control

The code shown in Listing 13.3, from `DataBindDataGrid1.aspx`, illustrates the binding of a `DataGridView` server control to a `DataView` object (in this case, the `DefaultView` of the first `DataTable` of the `DataSet`).

Listing 13.3 DataBindDataGrid1.aspx—Binding a DataGridView Control to a DataView Object

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

Listing 13.3 continued

```
<html>
<head>
<title>.DataBindDataGrid1.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub
Sub BindDataSet()
    ' Create the DataSet and bind to it
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String = _
        "SELECT EmployeeId, LastName, FirstName FROM Employees"

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    ' Create & execute SqlDataAdapter query to return records
    Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)
    ' Create new empty DataSet
    Dim ds As DataSet = New DataSet

    ' Fill DataSet using SqlDataAdapter
    sda.Fill(ds, "Employees")

    dgrEmployees.DataSource = ds.Tables(0).DefaultView
    ' The above is equivalent to:
    ' dgrEmployees.DataSource = ds
    dgrEmployees.DataBind()
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Northwind Employees

" />

    <asp:DataGrid id="dgrEmployees" runat="server"
        AutoGenerateColumns=True
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#99ccff"
```

Listing 13.3 continued

```

    AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>
```

Unlike the simpler DropDownList and TextBox controls used in earlier examples of this chapter, the DataGrid server control is a complex control that is a container for rows and columns of data. The DataGrid control, which was introduced in Chapter 10, “Designing Advanced Interfaces with Web Form List Controls,” allows you to control which columns are displayed and how those columns are displayed. In this example, however, the AutoGenerateColumns property of the DataGrid has been set to True, telling the DataGrid to generate a grid column for each of the columns supplied by the DataSet.

Don’t concern yourself too much with the code used to create the DataSet. This code will be explained in detail in Chapter 15. Once the DataSet is created, the critical code that performs the binding is shown here:

```
dgrEmployees.DataSource = ds.Tables(0).DefaultView
dgrEmployees.DataBind()
```

In this case, unlike in prior examples, the code is using the DataBind method of the DataGrid control rather than the Page object. In this example, you could have also used the Page object. It’s more efficient however to use an individual control’s DataBind method, rather than asking the Page object to scan the entire page for suitable data binding expressions to bind.

The DataBindDataGrid1.aspx page is shown in Figure 13.4.



Figure 13.4

Binding a DataSet to a DataGrid control.

TIP

Although the `DataSet` object does not directly support data binding, if you attempt to set the `DataSource` of a control to it, it will supply the `DefaultView` property of the first `DataTable` in the `DataSet`'s `Tables` collection for data binding.

The `DataBindDataGrid1.aspx` page serves as a second `DataGrid` example. In this case, the `AutoGenerateColumns` property of the `DataGrid` control is set to `False`. Individual columns are specified using `BoundColumn` controls as shown in Listing 13.4.

Listing 13.4 DataBindDataGrid1.aspx—This Code from DataBindDataGrid1.aspx Illustrates How to Bind to Specific BoundColumn Controls Contained Within a DataGrid Control

```
<asp:DataGrid id="dgrEmployees" runat="server"
    AutoGenerateColumns=False
    BorderColor="black"
    BorderWidth="1"
    GridLines="Both"
    CellPadding="3"
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#99ccff"
    AlternatingItemStyle-BackColor="lightgray" >
<Columns>
    <asp:BoundColumn HeaderText="Last Name" DataField="LastName" />
    <asp:BoundColumn HeaderText="First Name" DataField="FirstName" />
</Columns>
</asp:DataGrid>
```

Binding to a `DataSet` Control

Another of the list server controls introduced in Chapter 10 is the `DataSet` control. This control, along with the related `DataRepeater` control is known as a templated control because it allows you to finely tune the look of the control by supplying templates for its various parts.

The `DataBindDataList.aspx` page, shown in its entirety in Listing 13.5, demonstrates a `DataSet` control that is used to display images of books, along with descriptive text and hyperlinks, in a two-column snaking list.

Listing 13.5 DataBindDataList.aspx—This DataSet Control Displays a Photo, Text, and a Hyperlink for Each Item in a DataView of a DataSet

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

<html>
<head>
<title>DataBindDataList.aspx</title>
```

Listing 13.5 continued

```

<script language="VB" runat="server">
Sub Page_Load(Sender As Object, E As EventArgs)
    If Not Page.IsPostBack Then
        Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=pubs;"
        Dim strSQLEmp As String = "SELECT * FROM titles"

        Dim cnx As SqlConnection = New SqlConnection(strCnx)
        Dim sdaTitles As SqlDataAdapter = New SqlDataAdapter(strSQLEmp, cnx)
        Dim dsPubs As DataSet = New DataSet
        Dim drTitles As DataRow
        Dim dvTitles As DataView

        ' Fill Titles DataTable of dsPubs DataSet
        ' using the sdaTiles SqlDataAdapter
        sdaTitles.Fill(dsPubs, "Titles")

        ' A little trickery to add an important
        ' book to the Titles DataTable
        drTitles = dsPubs.Tables("Titles").NewRow
        drTitles("title_id") = "PC1000"
        drTitles("title") = "ASP.NET for Developers"
        drTitles("pub_id") = "1111"
        drTitles("type") = "popular_comp"
        drTitles("price") = "29.99"
        dsPubs.Tables("Titles").Rows.Add(drTitles)

        ' Create the dvTitles view so we can resort the
        ' rows by title (part of the trickery).
        dvTitles = New DataView(dsPubs.Tables("Titles"))
        dvTitles.Sort = "title"

        ' And bind the dvTitles view to the
        ' dlTitles DataList control
        dlTitles.DataSource = dvTitles
        dlTitles.DataBind()
    End If
End Sub
</script>
</head>
<body topmargin="0" leftmargin="0" marginwidth="0"
marginheight="0" width="100%">

<table width="100%" cellspacing="0" cellpadding="0">
    <tr>
        <td style="height:20" bgcolor="#9C0001"
cellspacing="0" cellpadding="0" />
    </tr>

```

Listing 13.5 continued

```
<tr>
  <td align="right"
    style="height:70;font-family:Arial;font-weight:bold;font-size:44pt;
    color:white"
    width="100%" bgcolor="#D3C9C7">BooksForU.com
  </td>
</tr>
</table>

<asp:DataList id="dlTitles" RepeatColumns="2" runat="server">
  <ItemTemplate>
    <table cellpadding="10" style="font: 10pt verdana">
      <tr>
        <td width="1" bgcolor="#BD8672"/>
        <td valign="top">
          <img align="top" src='<%# DataBinder.Eval(Container.DataItem, _ "title_id", "images/title-{0}.gif") %>' >
        </td>
        <td valign="top">
          <b>Title: </b><%# Container.DataItem("title") %><br>
          <b>Category: </b>
          <%# DataBinder.Eval(Container.DataItem, "type") %><br>
          <b>Publisher ID: </b>
          <%# DataBinder.Eval(Container.DataItem, "pub_id") %><br>
          <b>Price: </b>
          <%# DataBinder.Eval(Container.DataItem, "price", "$ {0}") %>
          <p>
            <a href='<%# DataBinder.Eval(Container.DataItem, _ "title_id", "purchase.aspx?titleid={0}") %>' >
              
            </a>
          </td>
        </tr>
      </table>
    </ItemTemplate>
  </asp:DataList>

  <table width="100%" cellspacing="0" cellpadding="0">
    <tr>
      <td style="height:20" bgcolor="#9C0001"
        cellspacing="0" cellpadding="0" width="100%"/>
    </tr>
  </table>

</body>
</html>
```

In the DataBindDataList.aspx example, the `ItemTemplate` code is called for every row returned by the `dvTitles` `DataView` object that is bound to the control. Within an `ItemTemplate` element, individual columns are bound to the HMTL using syntax like this for the `title` column:

```
<b>Title: </b><%# Container.DataItem("title") %><br>
```

The `Container.DataItem` method is used to tell ASP.NET to grab the data item (or column) from the object to which the `DataList` is bound—in this case, the `dvTitles` `DataView` object.

When using templated controls, you may have the need to format a string using the `DataBinder.Eval` method. This method evaluates data binding expressions at runtime and formats the output as a string. The basic syntax of `DataBinder.Eval` is

```
DataBinder.Eval(container, expression, formatstring)
```

`DataBinder.Eval` is employed several times in the `DataBindDataList.aspx` example (see Listing 13.5). For example, in the following code, `DataBinder.Eval` is used to format the `src` property of an `img` control:

```
<img align="top" src='<%# DataBinder.Eval(Container.DataItem, _  
"title_id", "images/title-{0}.gif") %>' >
```

The `DataBindDataList.aspx` page is shown in Figure 13.5.

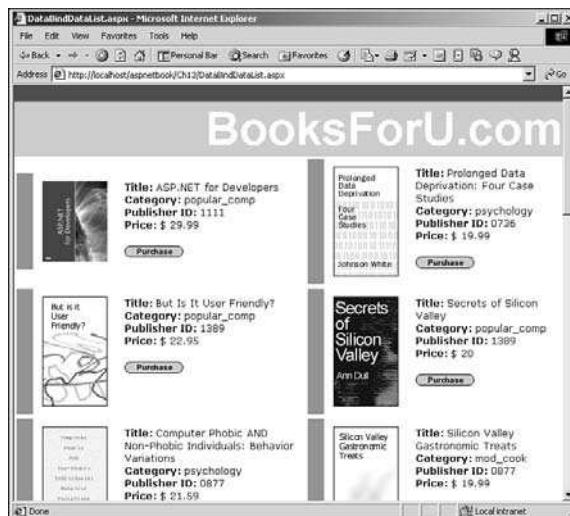


Figure 13.5

The `DataBindDataList.aspx` page displays publication titles along with images, text, and hyperlinks.

Summary

Data access in ASP.NET is provided by a set of classes known collectively as ADO.NET. These classes provide both database access and data containership. Database access is provided by the classes of the `System.Data.SqlClient` and `System.Data.OleDb` namespaces. The `System.Data.SqlClient` classes provide access to SQL Server databases via the Tabular Data Stream protocol. The `System.Data.OleDb` classes provide access to a great variety of databases using the OLE DB protocol. The `System.Data` namespace contains the `DataSet` class, a class which serves as an in-memory container for data.

Data binding in ASP.NET is robust and flexible. You can bind controls and HTML to properties, methods, functions, arrays, and collections supporting the `IEnumerable`, `ICollection`, or `IListSource` interfaces.

ADO.NET and data binding are discussed in further detail in Chapters 14 and 15.

CHAPTER 14

Accessing Data with .NET Data Providers

As mentioned in Chapter 13, “Introduction to ADO.NET,” ADO.NET contains two basic classes of objects: data container objects and data connectivity objects. In order to connect to a data source, you must use one of the ADO.NET .NET Data Providers of which (as this book went to press) there are two: the SQL Server data provider and the OLE DB data provider.

NOTE

The .NET Data Providers are also referred to as *.NET Managed Providers*.

In this chapter, you learn how to use .NET Data Providers to connect to, execute commands against, and retrieve and update data from relational databases.

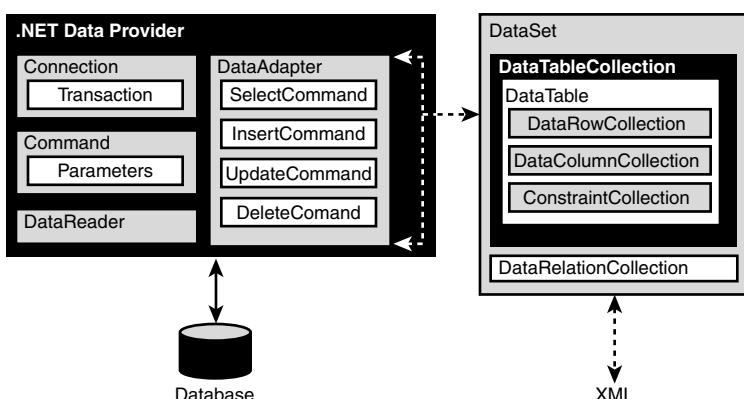
Working with .NET Data Providers

Beta 2 of the .NET Framework shipped with two .NET Data Providers for ADO.NET: the SQL Server and OLE DB data providers. Additional providers may become available later. The SQL Server provider works directly with Microsoft SQL Server databases via the `System.Data.SqlClient` classes. The OLE DB provider supports connectivity to OLE DB data sources via the `System.Data.OleDb` classes. Each of the .NET Data Providers provides core functionality exposed by four classes of objects: `Connection`, `Command`, `DataReader`, and `DataAdapter`. These core classes and the `SqlClient` and `OleDb` versions of these classes are summarized in Table 14.1.

Table 14.1 Core .NET Data Provider Classes

Generic Class	SqlClient Class	OleDb Class	Purpose
Connection	SqlConnection	OleDbConnection	Establishes connection to data source
Command	SqlCommand	OleDbCommand	Executes SQL statements against data source
DataReader	SqlDataReader	OleDbDataReader	Creates a read-only, forward-only stream of data from the data source
DataAdapter	SqlDataAdapter	OleDbDataAdapter	Provides a mechanism for moving records between the .NET Data Provider and a DataSet

The relationship between the .NET .NET Data Provider classes and the `DataSet` class is illustrated in Figure 14.1.

**Figure 14.1**

The relationship between ADO.NET .NET Data Providers and the `DataSet` object.

The SQL Server .NET Data Provider

The SQL Server .NET Data Provider is made up of the classes in the `System.Data.SqlClient` namespace. These classes include the `SqlConnection`, `SqlCommand`, `SqlDataReader`, and `SqlDataAdapter` classes.

You aren't precluded from using the OLE DB .NET Data Provider to access Microsoft SQL Server data, but in most cases you'll want to use the SQL Server .NET data provider which is optimized for access to SQL Server databases. The provider works directly with Microsoft SQL Server 7.0 and later using the SQL Server Tabular Data Stream (TDS), a highly optimized protocol for communicating with SQL Server.

If you need to access an older version of SQL Server, you will need to use the OLE DB .NET Data Provider along with the OLE DB Provider for SQL Server (SQLOLEDB).

The OLE DB .NET Data Provider

The OLE DB .NET Data Provider is made up of the classes in the `System.Data.OleDb` namespace. These classes include the `OleDbConnection`, `OleDbCommand`, `OleDbDataReader`, and `OleDbDataAdapter` classes.

The OLE DB .NET Data Provider accesses data via the OLE DB standard. OLE DB is a Microsoft-created open specification designed for accessing data from a variety of data sources via an OLE DB Data Provider. OLE DB providers are available for a wide variety of relational database products, including Microsoft Access, SQL Server, Oracle, and other relational database engines. OLE DB providers are also available for accessing non-relational data, including Microsoft Exchange, Active Directory Services, and other non-relational data stores. Microsoft also provides the Microsoft OLE DB Provider for ODBC Drivers, an OLE DB to Open Database Connectivity (ODBC) bridge that lets you access relational and pseudo-relational database engines using the older ODBC protocol.

Connecting to Data Using Connections

In order to connect to a data source, you must establish a data connection using a `Connection` class.

If you're using the SQL Server .NET Data Provider, you establish a data connection using the `SqlConnection` class. For example, the following code creates a connection to the SQL Server Northwind sample database on the local machine with a SQL Server user id of "sa" and a blank password:

```
Dim cnxSQLNWind As SqlConnection = _
    New SqlConnection("server=localhost;uid=sa;pwd=;database=northwind;")
```

The following code creates a connection to the pubs database on the server Coho with a user id of "mary" and a password of "cat":

```
Dim cnxPubs As SqlConnection = _
    New SqlConnection("server=coho;uid=mary;pwd=cat;database=pubs;")
```

The `OleDbConnection` class provides data connections for the OLE DB .NET Data Provider. For example, the following code creates a connection to the Microsoft Access Northwind sample database using the Microsoft Jet 4.0 OLE DB Data Provider. In this example, the database is located at c:\data\northwind.mdb and security has not been enabled:

```
Dim cnxAccessNWInd As OleDbConnection = _
    New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=c:\data\northwind.mdb;")
```

Before you can use a connection, you must open it using the `Open` method. For example:

```
cnxSQLNWind.Open()
```

TIP

There are some situations, such as when creating a `DataSet` using a `DataAdapter`, where ADO.NET will implicitly execute the `Open` method for you.

To close a connection, you use the `Close` method. For example:

```
cnxSQLNWind.Close()
```

WARNING

ADO.NET does not automatically drop a connection if the object variable goes out of scope. It's important to explicitly close connections using the `Close` method when you are done with a connection.

Executing SQL with Commands

You use the `Command` objects to execute SQL statements against a data source. Using `Command` objects, you can execute both ad-hoc SQL statements and stored procedures. You can execute SQL statements that return records, as well as SQL statements that do not return records. In this section, you'll look at the latter case—non-record returning queries. See “Fast Data Access with `DataReaders`” later in this chapter for details on executing queries that return records.

You use syntax similar to the following to execute ad-hoc SQL that doesn't return any records:

```
Dim command As SqlCommand|OleDbCommand = _  
    New SqlCommand|OleDbCommand (sqlstatement, openconnection)  
command.CommandType = CommandType.Text  
  
numrecs = command.ExecuteNonQuery()
```

Setting the `CommandType` of the `SqlCommand` or `OleDbCommand` object to `CommandType.Text` tells ADO.NET that you are passing it an ad-hoc SQL statement. You can also set `CommandType` to `CommandType.StoredProcedure` when executing a stored procedure or `CommandType.TableDirect` when passing the name of a table. `CommandType.TableDirect` is valid only when using the OLE DB .NET Data Provider.

You use the `ExecuteNonQuery` method of the `SqlCommand` or `OleDbCommand` object to execute a query that doesn't return records, such as a query that updates records or a query that creates database objects.

The `SqlCommandUpdate.aspx` page shown in Listing 14.1 executes one of two SQL `UPDATE` statements, depending upon which `Button` server control is clicked. The first `UPDATE` statement sets the `Region` field of Northwind `Customers` records to “Other” for records where `Region` is Null. The second `UPDATE` statement undoes this change, setting `Region` to Null where it's found to be “Other.”

Listing 14.1 SqlCommandUpdate.aspx—Executing SQL UPDATE Statements Against the Northwind SQL Server Database

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

<html>
<head>
<title>SqlCommandUpdate.aspx</title>
<script language="VB" runat="server">
Sub UpdateRegion(Src as Object, E as CommandEventArgs)
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String
    Dim intRecs As Integer
    ' Set SQL based on which button was clicked
    If E.CommandName = "Update" Then
        strSQL = "UPDATE Customers SET Region = 'Other' WHERE Region IS NULL"
    Else
        strSQL = "UPDATE Customers SET Region = NULL WHERE Region ='Other'"
    End If

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    ' Open the connection
    cnx.Open()

    ' Create & execute SqlCommand query to update records
    Dim cmdUpdate As SqlCommand = New SqlCommand(strSQL, cnx)
    cmdUpdate.CommandType = CommandType.Text

    ' Execute the query
    intRecs = cmdUpdate.ExecuteNonQuery()
    lblMsg.Text = intRecs.ToString() & " records updated!"

    ' Close the connection
    cnx.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <p>
        <asp:Label id="lblHeader" runat="server"
            Text="<h2>SqlCommandUpdate Example</h2>" />
    </p>
    <p>
        <asp:Button id="cmdUpdate" runat="server"
            Text="Update Null Regions to 'Other'"
            OnCommand="UpdateRegion"
            CommandName="Update" />
    </p>
</form>

```

Listing 14.1 continued

```
</p>
<p>
<asp:Button id="cmdReset" runat="server"
    Text="Reset 'Other' Regions to Null"
    OnCommand="UpdateRegion"
    CommandName="Reset" />
</p>
<p>
<asp:Label id="lblMsg" runat="server"
    Style="Color:red;" />
</p>
</form>
</body>
</html>
```

An analogous example that uses the OLE DB .NET Data Provider against the Microsoft Access Northwind database is shown in Listing 14.2.

Listing 14.2 OleDbCommandUpdate.aspx—Executing SQL UPDATE Statements Against the Northwind Access Database

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

<html>
<head>
<title>OleDbCommandUpdate.aspx</title>
<script language="VB" runat="server" src="AccessInclude.aspx">
    ' This include script contains the AccessCnx function
    ' which returns a connection string to point
    ' to the Access Northwind database.
    ' You may need to modify the connection string
    ' found in this include file.
</script>

<script language="VB" runat="server">
Sub UpdateRegion(Src as Object, E as CommandEventArgs)
    Dim strSQL As String
    Dim intRecs As Integer
    ' Set SQL based on which button was clicked
    If E.CommandName = "Update" Then
        strSQL = "UPDATE Customers SET Region = 'Other' WHERE Region IS NULL"
    Else
        strSQL = "UPDATE Customers SET Region = NULL WHERE Region ='Other'"
    End If

    ' Create connection to Access Northwind database
    Dim cnx As OleDbConnection = New OleDbConnection(AccessCnx())

```

Listing 14.2 continued

```

' Open the connection
cnx.Open()

' Create & execute OleDbCommand query to update records
Dim cmdUpdate As OleDbCommand = New OleDbCommand(strSQL, cnx)
cmdUpdate.CommandType = CommandType.Text

' Execute the query
intRecs = cmdUpdate.ExecuteNonQuery()
lblMsg.Text = intRecs.ToString() & " records updated!"

' Close the connection
cnx.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <p>
        <asp:Label id="lblHeader" runat="server"
            Text="

## OleDbCommandUpdate Example

" />
    </p>
    <p>
        <asp:Button id="cmdUpdate" runat="server"
            Text="Update Null Regions to 'Other'"
            OnCommand="UpdateRegion"
            CommandName="Update" />
    </p>
    <p>
        <asp:Button id="cmdReset" runat="server"
            Text="Reset 'Other' Regions to Null"
            OnCommand="UpdateRegion"
            CommandName="Reset" />
    </p>
    <p>
        <asp:Label id="lblMsg" runat="server"
            Style="Color:red;" />
    </p>
</form>
</body>
</html>
```

The OleDbCommandUpdate.aspx page found in Listing 14.2 uses an include file to provide the path to the Northwind.mdb database file. The include file, AccessInclude.aspx, is shown in Listing 14.3. Using this include file minimizes the impact of any changes to this file path. If you need to change the location of the Northwind.mdb file, you only need to change the path in this one file and all of the Access examples in this chapter will then employ the new path.

Listing 14.3 AccessInclude.aspx—Setting the Connection String for Connecting to the Microsoft Access Northwind Sample Database

```
Function AccessCnx() As String
    Return "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\Program Files\Microsoft Office" & _
    "\Office10\Samples\Northwind.mdb;"
End Function
```

Retrieving a Single Value

The `SqlCommand` and `OleDbCommand` objects contain a method, `ExecuteScalar`, that you can use to efficiently obtain a single value from a SQL statement without using a `DataSet`, `DataReader`, or a parameterized stored procedure. There's no restriction on the type of SQL statement you can use with `ExecuteScalar`. It simply returns the value of the first field in the first record returned by the query. All other fields and records are ignored.

The `SqlCommandScalar.aspx` page shown in Listing 14.4 demonstrates the `SqlCommand` object's `ExecuteScalar` method. The `DoCount` subroutine uses `ExecuteScalar` and a `SELECT COUNT(*)` query to return the number of Sales Representative employees at Northwind Traders.

Listing 14.4 SqlCommandScalar.aspx—Using the ExecuteScalar Method to Calculate the Number of Sales Representative Employees

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

<html>
<head>
<title>SqlCommandScalar.aspx</title>
<script language="VB" runat="server">
Sub DoCount(Src as Object, E as EventArgs)
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String =
        "SELECT COUNT(*) FROM Employees WHERE Title = 'Sales Representative'"
    Dim intCount As Integer

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    ' Open the connection
    cnx.Open()

    ' Create & execute SqlCommand query to update records
    Dim scdScalar As SqlCommand = New SqlCommand(strSQL, cnx)

    intCount = scdScalar.ExecuteScalar()

    lblMsg.Text = "Number of Sales Representatives in " & _
```

Listing 14.4 continued

```

"Northwind Employees table: " & _
intCount.ToString()

cnx.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
<p>
<asp:Label id="lblHeader" runat="server"
Text="<h2>SqlCommandScalar Example</h2>" />
</p>
<p>
<asp:Button id="cmdCount" runat="server"
Text="Count Sales Reps"
OnClick="DoCount" />
</p>
<p>
<asp:Label id="lblMsg" runat="server"
Style="Color:red; Font-size:12pt" />
</p>
</form>
</body>
</html>
```

In Listing 14.5, you'll find the `OleDbCommand` version of the `DoCount` subroutine from Listing 14.4. Except for the different class names and the different connection strings, the two examples are identical.

Listing 14.5 OleDbCommandScalar.aspx—The DoCount Subroutine from the OleDbCommand Version of the SqlCommandScalar.aspx Example

```

Sub DoCount(Src as Object, E as EventArgs)
    Dim strSQL As String = _
    "SELECT COUNT(*) FROM Employees WHERE Title = 'Sales Representative'"
    Dim intCount As Integer

    ' Create connection to SQL Server Northwind database
    Dim cnx As OleDbConnection = New OleDbConnection(AccessCnx())
    ' Open the connection
    cnx.Open()

    ' Create & execute OleDbCommand query to update records
    Dim scdScalar As OleDbCommand = New OleDbCommand(strSQL, cnx)

    intCount = scdScalar.ExecuteScalar()
```

Listing 14.5 continued

```
lblMsg.Text = "Number of Sales Representatives in " & _
    "Northwind Employees table: " & _
    intCount.ToString()

cnx.Close()
End Sub
```

Working with Stored Procedure Parameters

Stored procedures provide a number of distinct benefits over the use of ad-hoc SQL statements. These benefits include: better performance, greater reuse of code, and improved security, to name just a few. The `SqlCommand` and `OleDbCommand` classes both provide the ability to execute stored procedures and work with their parameters.

Executing a stored procedure containing parameters involves the following six steps:

1. Create a `SqlCommand` or `OleDbCommand` object, passing it the name of the stored procedure and a reference to an open `SqlConnection`/`OleDbConnection` as parameters.
2. Set the `CommandType` property of the `SqlCommand`/`OleDbCommand` object to `CommandType.StoredProcedure`.
3. Create a `SqlParameter`/`OleDbParameter` object for each input, output, or return value parameter.
4. Set the name, data type, direction, and value of each parameter.
5. Add the parameter to the `SqlCommand`/`OleDbCommand` object's parameters collection.
6. Use the appropriate `Execute` method (`ExecuteNonQuery`, `ExecuteScalar`, or `ExecuteReader`) to run the query.

The `SqlCommandSP.aspx` page, shown in Listing 14.6, employs a stored procedure to update the `UnitsOnOrder` and `UnitsOnStock` fields of the `Products` table in the SQL Server Northwind database.

Listing 14.6 SqlCommandSP.aspx—Executing a Parameterized Stored Procedure Using the SqlCommand and SqlParameter Objects

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

<html>
<head>
<title>SqlCommandUpdate.aspx</title>
<script language="VB" runat="server">
Sub CreateSP(cnxOpen)
    Dim strSQL As String =
        "CREATE PROCEDURE procReceiveStock " & _
        " @productid INTEGER, @units INTEGER AS " & _
```

Listing 14.6 continued

```

" IF (SELECT UnitsOnOrder FROM Products " & _
"     WHERE ProductId = @ProductId)>= @units " & _
" BEGIN " & _
"     UPDATE Products SET UnitsOnOrder = UnitsOnOrder - @units, " & _
"     UnitsInStock = UnitsInStock + @units " & _
"     WHERE ProductId = @productid " & _
"     RETURN 0 " & _
" END " & _
" ELSE " & _
"     RETURN 1 "

' Execute SqlCommand query to create stored proc
Try
    Dim cmdCreateSP As SqlCommand = New SqlCommand(strSQL, cnxOpen)
    cmdCreateSP.ExecuteNonQuery()
Catch
    ' Ignore error that will occur if stored procedure
    ' is already defined.
End Try
End Sub

Sub UpdateStock(Src as Object, E as CommandEventArgs)
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim cnx As SqlConnection
    Dim prm As SqlParameter
    Dim cmd As SqlCommand
    Dim intReturn as Integer

    ' Create connection to SQL Server Northwind database
    cnx = New SqlConnection(strCnx)
    ' Open the connection
    cnx.Open()

    ' Create stored procedure
    Call CreateSP(cnx)

    ' Create & execute SqlCommand query to update records
    cmd = New SqlCommand("procReceiveStock", cnx)
    cmd.CommandType = CommandType.StoredProcedure

    ' Create parameters
    prm = New SqlParameter("RETURN_VALUE", SqlDbType.Int)
    prm.Direction = ParameterDirection.ReturnValue
    cmd.Parameters.Add(prm)

    prm = New SqlParameter("@productid", SqlDbType.Int)
    prm.Direction = ParameterDirection.Input

```

Listing 14.6 continued

```
prm.Value = txtProductId.Text
cmd.Parameters.Add(prm)

prm = New SqlParameter("@units", SqlDbType.Int)
prm.Direction = ParameterDirection.Input
prm.Value = txtUnitsIn.Text
cmd.Parameters.Add(prm)

cmd.ExecuteNonQuery()

intReturn = cmd.Parameters("RETURN_VALUE").Value
If intReturn = 0 Then
    lblMsg.Text = "Record updated!"
Else
    lblMsg.Text = "Record not updated." & _
        "<br />Either there is no product with this Id or there " & _
        " is there are not that many units on order for this product."
End If

cnx.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <p>
        <asp:Label id="lblHeader" runat="server"
            Text="<h2>SqlCommandSP Example</h2>" />
    </p>
    <table>
        <tr>
            <td>Product Id:</td>
            <td><asp:Textbox id="txtProductId" runat="server" /></td>
        </tr>
        <tr>
            <td>Units of Stock Received:</td>
            <td><asp:Textbox id="txtUnitsIn" runat="server" /></td>
        </tr>
        <tr>
            <td><asp:Button id="cmdCreate" runat="server"
                Text="Update Stock " OnCommand="UpdateStock" /></td>
        </tr>
    </table>
    <p>
        <asp:Label id="lblMsg" runat="server"
            Style="Color:red;" />
    </p>
```

Listing 14.6 continued

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

The `UpdateStock` subroutine in `SqlCommandSP.aspx` employs `SqlCommand` and `SqlParameter` objects to populate the parameters of the `procReceiveStock` stored procedure. This stored procedure is not part of the Northwind sample database that ships with SQL Server, so before attempting to execute the stored procedure, `UpdateStock` calls the `CreateSP` subroutine to first create the stored procedure. (Obviously, you would not create the stored procedure on the fly like this in a production environment.)

The `CreateSP` subroutine is used to create the stored procedure using a `SqlCommand` object and the `ExecuteNonQuery` method. A `Try/Catch` statement is used to handle—and ignore—the error that would be generated if you attempt to create the stored procedure more than once:

```
Try
    Dim cmdCreateSP As SqlCommand = New SqlCommand(strSQL, cnxOpen)
    cmdCreateSP.ExecuteNonQuery()
Catch
    ' Ignore error that will occur if stored procedure
    ' is already defined.
End Try
End Sub
```

Once the stored procedure has been created, the `UpdateStock` subroutine of `SqlCommandSP.aspx` creates a `SqlCommand` object that it will use to execute the stored procedure:

```
cmd = New SqlCommand("procReceiveStock", cnx)
cmd.CommandType = CommandType.StoredProcedure
```

The first parameter—in this case, the return value of the stored procedure—is created and added to the `SqlCommand` object's `Parameters` collection with the following code:

```
' Create parameters
prm = New SqlParameter("RETURN_VALUE", SqlDbType.Int)
prm.Direction = ParameterDirection.ReturnValue
cmd.Parameters.Add(prm)
```

This code passes the `SqlParameter` constructor the name of the parameter—for the return value, you need to use the string “`RETURN_VALUE`”—and the datatype of the parameter. You need to set the datatype to one of the `SqlDbType` enumeration values, which can be `BigInt`, `Binary`, `Bit`, `Char`, `DateTime`, `Decimal`, `Float`, `Image`, `Int`, `Money`, `NChar`, `NText`, `NVarChar`, `Real`, `SmallDateTime`, `SmallInt`, `SmallMoney`, `Text`, `Timestamp`, `TinyInt`, `UniqueIdentifier`, `VarBinary`, `VarChar`, or `Variant`.

The `SqlParameter` object's `Direction` property is used to specify the type of parameter

using the `ParameterDirection` enumeration which can be `ReturnValue`, `Input`, `Output`, or `InputOutput`.

When creating an input parameter, you must also supply the `Value` property as shown in this code that defines the second parameter:

```
prm = New SqlParameter("@productid", SqlDbType.Int)
prm.Direction = ParameterDirection.Input
prm.Value = txtProductId.Text
cmd.Parameters.Add(prm)
```

Once the parameters are defined, the code in `UpdateStock` executes the query and retrieves the value of the return value parameter by its name using the following code:

```
cmd.ExecuteNonQuery()
intReturn = cmd.Parameters("RETURN_VALUE").Value
```

You use the same syntax to retrieve output parameters, replacing “`RETURN_VALUE`” with the name of the output parameter.

The `SqlCommandSP.aspx` page is shown in Figure 14.2.

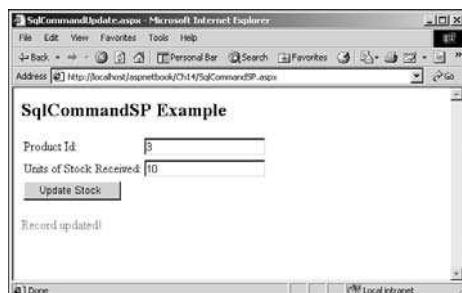


Figure 14.2

This page uses `SqlCommand` and `SqlParameter` objects to execute a stored procedure with parameters.

Fast Data Access with DataReaders

ADO.NET provides two sets of classes for retrieving rows from a database: `DataSet` and `DataReader`. The `DataSet` provides a rich, hierarchical data store. The `DataSet` class is navigable, supports multiple tables and relations, as well as updating of the data. In marked contrast, the `DataReader` classes—`SqlDataReader` and `OleDbDataReader`—provide very limited functionality. The `DataReaders` support the retrieval of rows in a forward-only, read-only manner. The `DataReaders` are perfect in those situations where you merely wish to retrieve a read-only set of data and plunk it into a `DataGridView` or similar control without the need for record navigation. If this is all you need to do with a set of data, the lean and mean `DataReader` classes are a wiser choice than the feature-rich but higher overhead `DataSet` class.

NOTE

See the section “Creating DataSets with the DataAdapters” later in this chapter and Chapter 15, “Working with ADO.NET DataSets,” for more on using the `DataSet` class.

Reading DataReader Rows

You retrieve data into a `DataReader` object using the `ExecuteReader` method of the appropriate `Command` object using syntax like this:

```
datareader = command.ExecuteReader()
```

To retrieve a row from a `DataReader`, you use the `Read` method:

```
datareader.Read()
```

The `Read` method advances the current row pointer and retrieves the record. `DataReaders`, by their nature, may only move forward through the rows. `Read` returns `True` if it successfully read a row or `False` if it is at the end of the `DataReader`. When you first create a `DataReader`, the current row pointer is positioned before the first record.

To loop through all the records in a `DataReader`, you use syntax similar to this:

```
Do While datareader.Read()
    ' process row
Loop
```

To read a value from the current row of a `DataReader`, you use the `Item` property using syntax like this:

```
datareader.Item("fieldname")
```

This can be shortened to:

```
datareader("fieldname")
```

The `OleDbDataReader.aspx` page shown in Listing 14.7 uses an `OleDbDataReader` to retrieve the values of `ProductName` and `UnitPrice` fields for each row in the `Products` table of the Access Northwind database.

Listing 14.7 OleDbDataReader.aspx—Using the Read Method to Read Records from a DataReader

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

<html>
<head>
<title>OleDbDataReader.aspx</title>
<script language="VB" runat="server" src="AccessInclude.aspx">
```

Listing 14.7 continued

```
' This include script contains the AccessCnx function
' which returns a connection string to point
' to the Access Northwind database.
' You may need to modify the connection string
' found in this include file.
</script>

<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    Dim strCnx As String = AccessCnx()
    Dim strSQL As String =
        "SELECT ProductName, UnitPrice FROM Products ORDER BY ProductName"

    ' Create connection to Access Northwind database
    Dim cnx As OleDbConnection = New OleDbConnection(strCnx)
    cnx.Open()

    ' Create & execute OleDbCommand query to return records
    Dim ocd As OleDbCommand = New OleDbCommand(strSQL, cnx)
    Dim odr As OleDbDataReader

    ' Fill DataReader using ExecuteReader method
    ' of the OleDbCommand object
    odr = ocd.ExecuteReader()

    ' Loop through DataReader records and display
    ' ProductName and UnitPrice fields in HTML table
    lblOut.Text &= "<table border=2>" &
        "<tr><th>Product</th><th>Price</th></tr>"
    Do While odr.Read()
        lblOut.Text &= "<tr><td>" & odr("ProductName") & "</td>" & _
            "<td align=right>" & FormatCurrency(odr("UnitPrice")) & "</td></tr>"
    Loop
    lblOut.Text &= "</table>

    ' Close DataReader and Connection
    odr.Close()
    cnx.Close()
End Sub
</script>
</head>

<body>
<form runat="server">
    <asp:Label id="lblHeader" runat="server"
        Text="<h2>Northwind Products</h2>" />
```

Listing 14.7 continued

```
<asp:Label id="lblOut" runat="server" />
</form>
</body>
</html>
```

The output generated by OleDbDataReader.aspx is shown in Figure 14.3.



A screenshot of Microsoft Internet Explorer displaying a table titled "Northwind Products". The table has two columns: "Product" and "Price". The data is as follows:

Product	Price
Alice Mutton	\$39.00
Aniseed Syrup	\$10.00
Boston Crab Meat	\$18.40
Camembert Fleurot	\$34.00
Carnarvon Tigers	\$62.50
Chai	\$18.00
Chang	\$19.00
Chartreuse verte	\$18.00
Chef Anton's Cajun Seasoning	\$22.00
Chef Anton's Gumbo Mix	\$21.35
Chocolate	\$12.75
Côte de Blaye	\$263.50
Escargots de Bourgogne	\$13.25
Fèves Mix	\$7.00
Fl阐明yost	\$21.50

Figure 14.3

This table was populated with data using an OleDbDataReader object and its Read method.

Using a DataReader with a DataGrid Control

You can bind a DataGrid control to a DataReader by setting the DataSource of the DataGrid control to the DataReader object using syntax like this:

```
datagridcontrol.DataSource = datareader
datagridcontrol.DataBind()
```

The SqlDataReaderGrid.aspx page shown in Listing 14.8 demonstrates binding a DataGrid control to a SqlDataReader.

Listing 14.8 SqlDataReaderGrid.aspx—This Page Employs a SqlDataReader Object to Populate a DataGrid Control with Rows from the SQL Server NorthWind Products Table

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

<html>
```

Listing 14.8 continued

```
<head>
<title>SqlDataReaderGrid.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub

Sub BindDataSet()
    ' Create the DataSet and bind to it
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String = "SELECT * FROM Products"

    ' Create connection to Access Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    cnx.Open()

    ' Create & execute OleDbCommand query to return records
    Dim scd As SqlCommand = New SqlCommand(strSQL, cnx)
    Dim sdr As SqlDataReader

    ' Fill DataReader using ExecuteReader method of the
    ' SqlCommand object.
    sdr = scd.ExecuteReader()

    ' Bind reader to grid
    dgrCustomers.DataSource = sdr
    dgrCustomers.DataBind()

    ' Close connection
    cnx.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Northwind Products

" />

    <asp:DataGrid id="dgrCustomers" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0" />
```

Listing 14.8 continued

```

Font-Name="Verdana"
Font-Size="8pt"
HeaderStyle-BackColor="#99ccff"
AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>

```

The page produced by the SqlDataReaderGrid.aspx page is shown in Figure 14.4.

ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	UnitsOnOrder	ReorderLevel	Discontinued
1	Chai	1	1	10 boxes x 20 bags	18	39	0	10	False
2	Chang	1	1	24 - 12 oz bottles	19	97	0	25	False
3	Aniseed Syrup	1	2	12 - 550 ml bottles	10	23	60	25	False
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22	53	0	0	False
5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.35	0	0	0	True
6	Greenline's Blackberry Spread	3	2	12 - 8 oz jars	25	120	0	25	False
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	30	15	0	10	False
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	40	6	0	0	False
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	97	29	0	6	True
10	Ikura	4	8	12 - 200 ml jars	31	31	0	0	False
11	Queso Cabrales	5	4	1 kg pkg.	21	22	30	30	False

Figure 14.4

This DataGrid control is bound to a SqlDataReader object.

NOTE

See Chapter 10, "Designing Advanced Interfaces with Web Form List Controls," for more details on using the DataGrid server control. See Chapter 13 for more information on data binding.

Creating DataSets with the DataAdapters

The DataAdapter classes—SqlDataAdapter and OleDbDataAdapter—bridge the .NET Data Provider classes with the DataSet class. The DataSet class is a feature-rich, navigable and hierarchical data store that is only briefly connected to a data source using the DataAdapter classes. There is no SqlDataAdapter or OleDbDataAdapter—the DataSet object works independently of its source of data. You use the appropriate DataAdapter class to connect the DataSet object to a data source to read or write records.

The procedure for creating a DataSet from database tables using a DataAdapter is as follows:

1. Create a new Connection object to the database.
2. Create a new DataAdapter object and pass it a query.
3. Create a new DataSet object.
4. Execute the Fill method of the DataAdapter object to fill a DataTable in the DataSet with the data returned by the DataAdapter's query.

The code in Listing 14.9, from SqlDataAdapterGrid.aspx, creates a DataSet from the SQL Server Northwind sample database using a SqlDataAdapter object.

***Listing 14.9 SqlDataAdapterGrid.aspx—Using a DataSet and the
SqlDataAdapter to Display SQL Server Northwind Database Customer
Records in a DataGrid***

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

<html>
<head>
<title>SqlDataAdapterGrid.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub
Sub BindDataSet()
    ' Create the DataSet and bind to it
    dgrCustomers.DataSource = CreateDataSet()
    dgrCustomers.DataBind()
End Sub
Function CreateDataSet() As DataSet
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String = "SELECT * FROM Customers"

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    ' Create & execute SqlDataAdapter query to return records
    Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)
    ' Create new empty DataSet
    Dim ds As DataSet = New DataSet

    ' Fill DataSet using SqlDataAdapter
    sda.Fill(ds, "Customers")
    Return ds
End Function
Sub HandlePaging(sender As Object, e As DataGridPageChangedEventArgs)
    ' Set CurrentPageIndex to the desired page

```

Listing 14.9 continued

```

dgrCustomers.CurrentPageIndex = e.NewPageIndex
Call BindDataSet()
End Sub
</script>
</head>
<body>
<form runat="server">
<asp:Label id="lblOut" runat="server"
Text="

## Northwind Customers

" />

<asp:DataGrid id="dgrCustomers" runat="server"
BorderColor="black"
BorderWidth="1"
GridLines="Both"
CellPadding="3"
CellSpacing="0"
Font-Name="Verdana"
Font-Size="8pt"
AllowPaging="True"
PageSize="10"
PagerStyle-Mode="NumericPages"
PagerStyle-HorizontalAlign="Right"
OnPageIndexChanged="HandlePaging"
HeaderStyle-BackColor="#99ccff"
AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>

```

The DataSet is created in the CreateDataSet function of Listing 14.9. This example uses the SQL Server .NET Data Provider's SqlConnection and SqlDataAdapter objects to connect to and retrieve records from the Customers table:

```

Function CreateDataSet() As DataSet
Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
Dim strSQL As String = "SELECT * FROM Customers"

' Create connection to SQL Server Northwind database
Dim cnx As SqlConnection = New SqlConnection(strCnx)
' Create & execute SqlDataAdapter query to return records
Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)

```


NOTE

If you wished to use the OleDbConnection and OleDbDataAdapter objects instead of the SqlConnection and SqlDataAdapter objects, the code would be virtually identical except for changes to the connection string and object names.

Next, an empty DataSet object, `ds`, is created with this code:

```
' Create new empty DataSet  
Dim ds As DataSet = New DataSet
```

The `Fill` method of the `SqlDataAdapter` object is used to fill the `DataSet` (`ds`) with the records from the “`SELECT * FROM Customers`” query, placing the records returned by the `SqlDataAdapter`’s query into a `DataTable` named `Customers`:

```
' Fill DataSet using SqlDataAdapter  
sda.Fill(ds, "Customers")
```

Because `DataSets` in ADO.NET can contain multiple `DataTable` objects, you must specify the name of the `DataTable` that will contain the records as the second parameter of the `Fill` method. The fact that the name of the `DataTable` used in this example (`Customers`) is the same as the name of the original database table is purely coincidental.

Finally, the `CreateDataSet` function returns the `ds` `DataSet` in its return value:

```
Return ds
```

The `GridView` in `SqlDataAdapterGrid.aspx` is bound to the `DataSet` using the following subroutine from Listing 14.9:

```
Sub BindDataSet()  
    ' Create the DataSet and bind to it  
    dgrCustomers.DataSource = CreateDataSet()  
    dgrCustomers.DataBind()  
End Sub
```

The `BindDataSet` subroutine sets the `DataSource` property of the `dgrCustomers` `GridView` control to the return value of `CreateDataSet`. Recall that `CreateDataSet` returns a `DataSet`. The `dgrCustomers` `GridView` is then bound to the `DataSet` using the `DataBind` method.

NOTE

For more information on data binding, see Chapter 13. For more on working with `GridViews`, see Chapter 10.

The `SqlDataAdapterGrid.aspx` example can be seen in Figure 14.5.

The screenshot shows a Microsoft Internet Explorer window with the title "SqlDataAdapterGrid.aspx - Microsoft Internet Explorer". The address bar shows the URL "http://localhost/intranetbook/Ch14/SqldataAdapterGrid.aspx". The main content area displays a DataGrid titled "Northwind Customers". The DataGrid has columns for CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, and Fax. The data includes records for various companies like ALFKI, ANATR, ANTON, AROUT, BERGS, BLAUS, BLOWP, BOLID, BONAP, and BOTTM. At the bottom of the grid, there is a navigation bar with buttons labeled 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11.

CustomerID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
ALFKI	Abercrombie & Fitch	Maria Anders	Sales Representative	Obere Str. 57	Berlin		12209	Germany	(030) 0074321	(030) 0976545
ANATR	Antonio Moreno Taquería	Ana Trujillo	Owner	Ave. de la Constitución 2222	México D.F.		05021	Mexico	(5) 555-4729	(5) 555-3745
ANTON	Antonio Moreno Taquería	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) 555-0700	(171) 555-0700	
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå		9-958 22	Sweden	0921-12 34	0921-12 34
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim		68206	Germany	0621-08460	0621-08924
BLOWP	Blondordstörlit père et fils Citeaux	Friedérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg		67000	France	88.60.15.31	88.60.15.32
BOLID	Bólido Comidas preparadas	Marin Sommer	Owner	C/ Aragual, 67	Madrid		28023	Spain	(91) 555 22 82	(91) 555 91 99
BONAP	Bon app'	Laurence Delteilan	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 14.5

This DataGrid was created using the DataSet and SqlDataAdapter classes.

You also use the DataAdapter classes when writing DataSet records back to the data source. This topic is discussed in Chapter 15.

Summary

ADO.NET provides two .NET Data Providers that you can use to read and write rows of database data. In this chapter you examined the differences between the SQL Server and OLE DB Data Providers.

You learned how to use the SqlConnection and OleDbConnection classes to connect to data sources. You saw how to use the SqlCommand and OleDbCommand classes to execute SQL statements and stored procedures. You also looked at how to use these classes along with the SqlDataReader and OleDbDataReader classes to efficiently retrieve query results and display them on an ASP.NET page. Finally, you took a look at using the SqlDataAdapter and OleDbDataAdapter classes to load data into a DataSet.

In the next chapter, you'll dig deeper into the DataSet class and learn how to use this versatile class to manipulate hierarchical sets of data, read and write XML, and update database data.

CHAPTER 15

Working with ADO.NET DataSets

An ADO.NET `DataSet` is in-memory representation of a set of data. `DataSets` are pretty versatile and may contain multiple tables, views, and relationships between the tables. `DataSets` work independently of the original source of the data. In fact, you can't connect `DataSets` directly to a database or other source of data. You can, however, use a `DataAdapter` object to move data between a `DataSet` and a data source.

`DataSets` were introduced briefly in Chapter 14, “Accessing Data with .NET Data Providers,” which discussed how to create a basic `DataSet` from a database query. In this chapter, you’ll dig deeper into how `DataSets` work and how to use them to retrieve, work with, and update database data. You’ll explore the child objects of a `DataSet`, including the `DataTable`, `DataView`, and `DataRelation` objects. You’ll also learn how to create a `DataSet` programmatically from code as well as from an XML file. Finally, you’ll learn how to take a `DataSet` and generate XML from it.

NOTE

If you’ve used ADO before, you might see some similarities between the ADO.NET `DataSet` and the ADO `Recordset`. There are a number of similarities but also some major differences. Both `Recordsets` and `DataSets` store tabular data that you can navigate and update, as well as bind to. Unlike `Recordsets`, however, `DataSets` work in a disconnected mode. They don’t know anything about the data source from which they came. Also, unlike the simpler `Recordset`, a `DataSet` can contain multiple tables, views, and relations. Finally, `DataSets` have more complete support for reading and writing XML.

Creating a DataSet

There are several ways to create a `DataSet`. You can

- load data from database tables
- load data from XML
- programmatically create and populate the `DataSet` using code

In many cases, you will want to create a `DataSet` from existing database tables. This technique was discussed in Chapter 14, in the section entitled "Reading and Writing DataSets with the DataAdapters." The next few sections of this chapter will continue to work with `DataSets` generated from database data. Later sections of the chapter will discuss creating `DataSets` using the latter two techniques.

Working with DataTables

The `DataSet` object contains a collection of `DataTable` objects. You access a `DataSet`'s `DataTable` objects via its `Tables` property:

```
dataset.Tables("data_table_name")
```

For example, the following code creates a new `DataTable` object and points it to `Orders` `DataTable` of the `dsPubs` `DataSet`:

```
Dim dtOrders As DataTable  
dtOrders = dsPubs.Tables("Orders")
```

You can also refer to a `DataTable` by its ordinal position in the collection of `DataTable`s. For example, if the `Orders` `DataTable` was the first `DataTable` in the `DataTable`s collection of the `dsPubs` `DataSet`, you could use the following code to get the same effect:

```
Dim dtOrders As DataTable  
dtOrders = dsPubs.Tables(0)
```

The code in Listing 15.1, from `DataTables.aspx`, creates a `DataSet` containing two `DataTable`s, `Customers`, and `Employees` which are used to populate two `DropDownList` controls.

Listing 15.1 DataTables.aspx—The Two DropDownList Controls on This Page Are Supplied by DataTables from a Single DataSet

```
<%@ Page Language="VB" Debug="true" %>  
<%@ Import Namespace="System.Data" %>  
<%@ Import Namespace="System.Data.SqlClient" %>  
  
<html>  
<head>  
<title>DataTables.aspx (SqlClient)</title>  
<script language="VB" runat="server">  
Sub Page_Load(Src As Object, E As EventArgs)
```

Listing 15.1 continued

```

If Not Page.IsPostBack Then
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQLCust As String = _
        "SELECT CustomerId, CompanyName FROM Customers ORDER BY CompanyName"
    Dim strSQLEmp As String = _
        "SELECT EmployeeId, LastName + ', ' + FirstName AS EmployeeName " & _
        "FROM Employees ORDER BY LastName, FirstName"

    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    Dim sdaCust As SqlDataAdapter = New SqlDataAdapter(strSQLCust, cnx)
    Dim sdaEmp As SqlDataAdapter = New SqlDataAdapter(strSQLEmp, cnx)
    Dim dsNWind As DataSet = New DataSet
    Dim dtCust As DataTable
    Dim dtEmp As DataTable

    sdaCust.Fill(dsNWind, "Customers")
    sdaEmp.Fill(dsNWind, "Employees")

    dtCust = dsNWind.Tables("Customers")
    dtEmp = dsNWind.Tables("Employees")

    drpCustomers.DataSource = dtCust.DefaultView
    drpCustomers.DataTextField = "CompanyName"
    drpCustomers.DataValueField = "CustomerId"
    drpCustomers.DataBind()

    drpEmployees.DataSource = dtEmp.DefaultView
    drpEmployees.DataTextField = "EmployeeName"
    drpEmployees.DataValueField = "EmployeeId"
    drpEmployees.DataBind()

End If
End Sub
Sub DisplayCust(Src as Object, E as EventArgs)
    lblCustPick.Text = "You have selected Customer " & _
        drpCustomers.SelectedItem.Value & "."
End Sub
Sub DisplayEmp(Src as Object, E as EventArgs)
    lblEmpPick.Text = "You have selected Employee " & _
        drpEmployees.SelectedItem.Value & "."
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblTitle" runat="server" Text="<h2>Northwind DataTables Example</h2>" />
    <p>Customers: <asp:DropDownList id="drpCustomers" 

```

Listing 15.1 continued

```
        runat="server" AutoPostBack=true  
        OnSelectedIndexChanged="DisplayCust" />  
<asp:Label id="lblCustPick" runat="server"  
        style="Color:red" /></p>  
<p>Employees: <asp:DropDownList id="drpEmployees"  
        runat="server" AutoPostBack=true  
        OnSelectedIndexChanged="DisplayEmp" />  
<asp:Label id="lblEmpPick" runat="server"  
        style="Color:red"/></p>  
</form>  
</body>  
</html>
```

The following code from Listing 15.1 creates the two `DataTable` objects:

```
Dim dtCust As DataTable  
Dim dtEmp As DataTable
```

This code sets the empty `DataTable` objects to point to the `DataSet`'s `DataTables`:

```
dtCust = dsNWind.Tables("Customers")  
dtEmp = dsNWind.Tables("Employees")
```

Each `DropDownList` control is bound to the `DefaultView` property of the appropriate `DataTable` using the following code:

```
drpCustomers.DataSource = dtCust.DefaultView  
drpCustomers.DataTextField = "CompanyName"  
drpCustomers.DataValueField = "CustomerId"  
drpCustomers.DataBind()  
  
drpEmployees.DataSource = dtEmp.DefaultView  
drpEmployees.DataTextField = "EmployeeName"  
drpEmployees.DataValueField= "EmployeeId"  
drpEmployees.DataBind()
```

Manipulating Rows of Data

The example shown in Listing 15.1 uses data binding to bind the data from the `DataTables` to `DropDownList` server controls. There may be times, however, where you'd like to directly manipulate the rows of a `DataTable`. You can use the `Rows` property of a `DataTable` to access the `DataTable`'s collection of `DataRow`s.

Typically, when working with `DataRow`s, you'll want to walk through each of the `DataRow`s within a `DataTable` with code similar to this:

```
For Each datarow In datatable.Rows  
    ' Process the row  
Next
```

To retrieve the *i*th row in the `DataRow`s collection you can use code like this:

```
datatable.Rows(i)
```

TIP

Like all ADO.NET collections, the `DataRow`s collection is a zero-based collection. Thus the first `DataRow` in the collection is row 0.

Counting Rows

The `Count` property of the `DataRow`s collection returns the number of rows in the `DataTable`. For example:

```
numrows = datatable.Rows.Count
```

Retrieving Field Values

You use the `Item` property of a `DataRow` to retrieve the value of a field for a particular row. For example, the following code retrieves the value of the `OrderDate` field for each of the rows in the `dtOrder` `DataTable`:

```
For Each drOrder In dtOrder.Rows
    Response.Write(drOrder.Item("OrderDate") & "<br />")
Next
```

The `Item` property is assumed if you don't specify it, so the previous code could be shortened to

```
For Each drOrder In dtOrder.Rows
    Response.Write(drOrder("OrderDate") & "<br />")
Next
```

The `DataRow.aspx` page, shown in Listing 15.2, walks through the rows in the `Customers` `DataTable` of the `dsNorthwind` `DataSet`, retrieving the value of the `CustomerName` field for each row and appending it to the `Text` property of the `Label` control. The code in Listing 15.2 also displays the number of rows in the `DataTable` using the `Rows.Count` property.

Listing 15.2 DataRow.aspx—Walking Through the Rows of the Customers DataTable

```
<html>
<head>
<title>DataRows.aspx (SqlClient)</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String = "SELECT * FROM Customers"

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
```

Listing 15.2 continued

```

' Create & execute SqlDataAdapter query to return records
Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)
' Create new empty DataSet
Dim dsNWind As DataSet = New DataSet
Dim dtCust as DataTable
Dim drCust As DataRow

' Fill DataSet using SqlDataAdapter
sda.Fill(dsNWind, "Customers")

' Set dtCust to point to Customers DataRow
dtCust = dsNWind.Tables("Customers")

' Display the row count
lblOut.Text &= dtCust.Rows.Count & " rows." & "  
" & "  
"

' Loop through the Rows collection of
' the DataTable and display the value of the
' CompanyName field for each DataRow
For Each drCust In dtCust.Rows
    lblOut.Text &= drCust("CompanyName") & "  
"
Next
End Sub
</script>
</head>
<body>
<asp:Label id="lblOut" runat="server"
    Text="

## Northwind Companies

" />
</body>
</html>

```

The `DataRow.aspx` page is shown in Figure 15.1.



Figure 15.1

This page walks through the rows of the Customers DataTable.

Filtering, Sorting, and Binding with DataViews

Most SQL databases support the creation of *views*—filtered views of data. Similarly, DataSets also support the idea of views, via the `DataView` object. A `DataView` is a view of a `DataTable` to which you can apply a filter or sort. Perhaps even more importantly, it is the `DataView` object that supports data binding for a `DataSet`.

NOTE

There are a few differences between the SQL View and the ASP.NET `DataView` worth noting: Unlike SQL Views, `DataViews` may be based on single table only. `DataViews` may be sorted, while SQL Views do not support sorting.

The `DefaultView` Property

Every `DataTable` in a `DataSet` has a `DefaultView` property that returns a default unsorted, unfiltered view of the data for the purposes of data binding.

The following code from the earlier Listing 15.1 sets the `DataSource` property of a `DropDownList` control to the `DefaultView` of the `dtCust` `DataTable`:

```
drpCustomers.DataSource = dtCust.DefaultView
```

ASP.NET allows you to directly bind to a `DataSet` using syntax like this:

```
control.DataSource = dataset
```

This works, however, only because you are really binding the `DefaultView` of the first `DataTable` in the `DataSet` to the control's `DataSource`. In fact, the preceding is really just a shortcut for the more complete code shown here:

```
control.DataSource = dataset.Tables(0).DefaultView
```

Thus, the following two lines of code are equivalent:

```
dgrCustomers.DataSource = dsNWInd
dgrCustomers.DataSource = dsNWInd.Tables(0).DefaultView
```

Creating Custom Views

Using `DataViews`, you can create two or more views of the same `DataTable` that filter or order the data in completely different ways. To create a new `DataView`, you use syntax like this:

```
Dim dataview As DataView = New DataView(datatable)
dataview.RowFilter = filtering_expression
dataview.Sort = sorting_expression
```

Of course, you don't have to sort and filter the data. In fact, you don't have to do either, but there's not much sense in creating a custom `DataView` if you're not going to rework the data in some way.

The filtering expression needs to be in form:

field operator value

For example, to create a RowFilter for the dvCustomer DataView that returned all rows where Region, a Text field, was “CA,” you would use code similar to the following:

```
dvCustomer.RowFilter = "Region = 'CA'"
```

To create a RowFilter for the dvEmployee DataView that returned all rows where EmployeeId, an Integer field, was 6, you would use code like this:

```
dvEmployee.RowFilter = "EmployeeId = 6"
```

The sorting expression needs to be in the form:

field1 ASC|DESC, field2 ASC|DESC, ...

Include the keyword ASC to sort rows in ascending order (the default if unspecified) or DESC to sort rows in descending order.

For example, to sort the dbEmployee DataView in ascending order by LastName and FirstName, you might use:

```
dvEmployee.Sort = "LastName, FirstName"
```

This is equivalent to

```
dvEmployee.Sort = "LastName ASC, FirstName ASC"
```

To sort the dvOrder DataView in descending date by OrderDate and ascending order by CompanyName, you might use

```
dvOrder.Sort = "OrderDate DESC, CompanyName ASC"
```

The DataViews.aspx page, shown in Listing 15.3, creates two cascading DropDownList controls. If you choose a country using the first DropDownList control, the FilterCountry event handler filters the second DropDownList so that it displays Customers in that country.

Listing 15.3 DataViews.aspx—Using a DataView to Create a Set of Cascading DropDownList Controls

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

<html>
<head>
<title>DataViews.aspx</title>
<script language="VB" runat="server">
```

Listing 15.3 continued

```
Sub Page_Load(Src As Object, E As EventArgs)
If Not Page.IsPostBack Then
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQLCust As String = _
        "SELECT CustomerId, CompanyName, Country " & _
        "FROM Customers ORDER BY CompanyName"
    Dim strSQLCountry As String = _
        "SELECT DISTINCT Country FROM Customers ORDER BY Country"
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    Dim sdaCust As SqlDataAdapter = New SqlDataAdapter(strSQLCust, cnx)
    Dim sdaCountry As SqlDataAdapter = New SqlDataAdapter(strSQLCountry, cnx)
    Dim dsNWInd As DataSet = New DataSet
    Dim dvCountryCust As DataView

    ' Fill Customers DataTable
    sdaCust.Fill(dsNWInd, "Customers")
    ' Fill Countries DataTable
    sdaCountry.Fill(dsNWInd, "Countries")

    ' Bind Countries DataTable to drpCountry DropDownList control
    drpCountry.DataSource = dsNWInd.Tables("Countries")
    drpCountry.DataTextField = "Country"
    drpCountry.DataValueField= "Country"
    drpCountry.DataBind()

    ' Initially disable drpCountryCust control
    drpCountryCust.Enabled = False
End If
End Sub

Sub FilterCountry(Src as Object, E as EventArgs)
    Dim dsNWInd As DataSet
    Dim dvCountryCust As DataView

    ' Retrieve dsNWInd from data cache
    dsNWInd = Cache("dsNWInd")

    ' Create new DataView using selection from drpCountry control
    dvCountryCust = New DataView(dsNWInd.Tables("Customers"))
    dvCountryCust.RowFilter = "Country = '" & _
        drpCountry.SelectedItem.Value & "'"
    dvCountryCust.Sort = "CompanyName"
```

Listing 15.3 continued

```
' Enable drpCountryCust control and bind the DataView to it
drpCountryCust.Enabled = True
drpCountryCust.DataSource = dvCountryCust
drpCountryCust.DataTextField = "CompanyName"
drpCountryCust.DataValueField = "CustomerId"
drpCountryCust.DataBind()
End Sub

Sub DisplayCountryCustomers(Src as Object, E as EventArgs)
    lblCountryCustPick.Text = "You have selected Customer " & _
        drpCountryCust.SelectedItem.Value & "."
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblTitle" runat="server"
        Text="

## Northwind CascadingDropDowns Example

" />
    <p>Country: <asp:DropDownList id="drpCountry"
        runat="server" AutoPostBack=true
        OnSelectedIndexChanged="FilterCountry" /></p>
    <p>Customers filtered by country: <asp:DropDownList id="drpCountryCust"
        runat="server" AutoPostBack=true
        OnSelectedIndexChanged="DisplayCountryCustomers" /></p>
    <asp:Label id="lblCountryCustPick" runat="server"
        style="Color:red"/></p>
</form>
</body>
</html>
```

The `DataViews.aspx` page makes use of the ASP.NET Data Cache to persist the `DataSet` between postbacks.

NOTE

For more information on caching, see Chapter 19, “Tracing, Debugging, and Optimizing Your ASP.NET Applications.”

The `DataViews.aspx` page is shown in Figure 15.2.

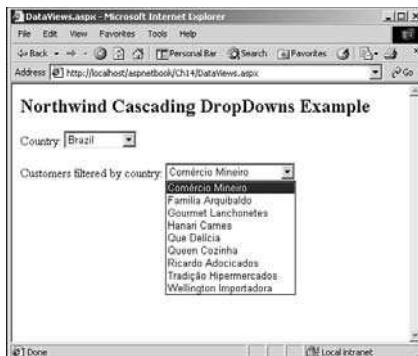


Figure 15.2

These DropDownList controls are *linked* so that when *Brazil* is selected in the first control, only Brazilian customers are shown in the second DropDownList control.

Relating Tables with the DataRelation Object

One of the nice things about DataSets is the ability to create relationships by virtue of the DataRelation object.

To create a DataRelation you use syntax like the following:

```
Dim datarelation As DataRelation = _
dataset.Relations.Add("datarelationname", _
dataset.Tables("parentdatatable").Columns("primarykey"), _
dataset.Tables("childdatatable").Columns("foreignkey"))
```

Once created, you can use a DataRelation to walk the child rows related to a row of the parent table using syntax like this:

```
For Each childdatarow In parentdatarow.GetChildRows(dataset.Relations
➥("datarelationname"))
    ' Process childdatarow
Next
```

The DataRelation.aspx page shown in Listing 15.4 demonstrates creating a DataRelation between the Customers and Orders DataTables of the dsNWInd DataSet.

Listing 15.4 DataRelation.aspx—Establishing a DataRelation Between the Customers and Orders Tables of the Northwind Sample

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

Listing 15.4 continued

```
<html>
<head>
<title>DataRelation.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    If Not Page.IsPostBack Then
        Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
        Dim strSQLCustomers As String = _
            "SELECT * FROM Customers ORDER BY CompanyName"
        Dim strSQLOrders As String = "SELECT * FROM Orders"
        Dim cnx As SqlConnection = New SqlConnection(strCnx)
        Dim sdaCustomers As SqlDataAdapter = _
            New SqlDataAdapter(strSQLCustomers, cnx)
        Dim sdaOrders As SqlDataAdapter = New SqlDataAdapter(strSQLOrders, cnx)
        Dim dsNWind As DataSet
        Dim drCustomer As DataRow
        Dim drOrder As DataRow
        Dim drlCustOrder As DataRelation

        dsNWind = New DataSet

        sdaCustomers.Fill(dsNWind, "Customers")
        sdaOrders.Fill(dsNWind, "Orders")

        ' Create relation between Customers and Orders
        drlCustOrder = dsNWind.Relations.Add("CustOrders", _
            dsNWind.Tables("Customers").Columns("CustomerId"), _
            dsNWind.Tables("Orders").Columns("CustomerId"))

        ' Iterate over data of customers and their orders
        ' First grab a Customer row
        For Each drCustomer In dsNWind.Tables("Customers").Rows
            lblOut.Text &= "<p><b>Customer</b>: " & drCustomer("CompanyName")
            lblOut.Text &= "<table border bgcolor=lightgreen>"
            lblOut.Text &= "<tr><th>OrderId</th> <th>Order Date</th></tr>" 
            ' Now grab each related Order row for the current Customer row
            For Each drOrder In _
                drCustomer.GetChildRows(dsNWind.Relations("CustOrders"))
                lblOut.Text &= _
                    "<tr><td> " & Convert.ToString(drOrder("OrderId")) & "</td>" 
                lblOut.Text &= _
                    "<td> " & Convert.ToString(drOrder("OrderDate")) & "</td></tr>" 
            Next
            lblOut.Text &= "</table></p>"
        Next
    End If
End Sub
```

Listing 15.4 continued

```
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Northwind Relations Example

" />
</form>
</body>
</html>
```

The DataRelation.aspx page is shown in Figure 15.3.

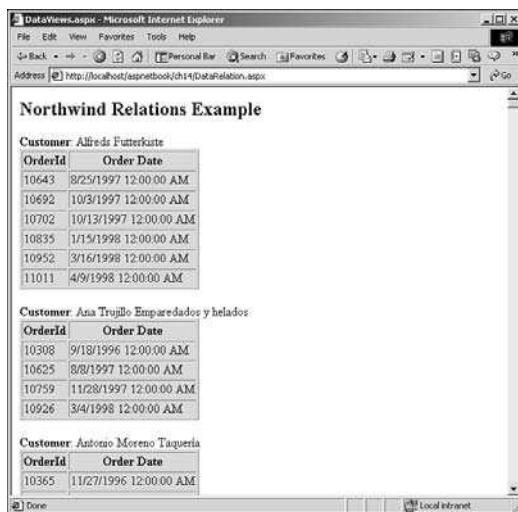


Figure 15.3

This page displays customers and their orders using a DataRelation.

Fabricating DataSets

There may be times when you wish to create a **DataSet** out of thin air. That is, when you wish to create a **DataSet** from code alone without any connection to a data source or XML file. ADO.NET supports the fabrication of **DataSets** and **DataTables**.

The basic steps to fabricating a **DataSet** containing a single **DataTable** are

1. Create a new **DataSet**.
2. Create a new **DataTable**.
3. Add columns to the **DataTable**. To add a column, create a new **DataColumn** object, set properties of the **DataColumn**, and append it to the **Columns** collection of the **DataTable** using the **Add** method.

4. Designate the primary key for the DataTable. To accomplish this, create a new array of type DataColumn, add the columns making up the primary key to the array, and set the PrimaryKey property of the DataTable to the array.
5. Add the DataTable to the Tables collection of the DataSet.
6. Populate the DataTable with rows.

If the DataSet is to contain multiple DataTables, repeat Steps 2 through 5 for each additional table.

These steps are illustrated by the DataSetFab.aspx example shown in Listing 15.5.

Listing 15.5 DataSetFab.aspx—Fabricating a DataSet

```
<%@ Page Language="VB" Debug="true" %>
<%@ Import Namespace="System.Data" %>

<html>
<head>
<title>DataSetFab.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub

Sub BindDataSet()
    ' Create the DataSet and bind to it
    dgrPeople.DataSource = FabricateDataSet()
    dgrPeople.DataBind()
End Sub

Function FabricateDataSet() As DataView
    ' Create new empty DataSet
    Dim dsFab As DataSet = New DataSet
    Dim dtPeople As DataTable = New DataTable
    Dim dc As DataColumn
    Dim drPeople As DataRow

    ' Create the PersonId Column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.Int32")
        .ColumnName = "PersonId"
        .Unique = True
        .AutoIncrement = True
        .AutoIncrementSeed = 101
        .AutoIncrementStep = 1
    End With
    dtPeople.Columns.Add(dc)
    dsFab.Tables.Add(dtPeople)
    Return dsFab
End Function
```

Listing 15.5 continued

```
End With
dtPeople.Columns.Add(dc)

' Create the LastName column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "LastName"
End With
dtPeople.Columns.Add(dc)

' Create the FirstName column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "FirstName"
End With
dtPeople.Columns.Add(dc)

' Create the EMail column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "Email"
End With
dtPeople.Columns.Add(dc)

' Make PersonId the primary key of the DataTable
Dim dcPk() As DataColumn = {dtPeople.Columns("PersonId")}
dtPeople.PrimaryKey = dcPk

' Add the dtPeople DataTable to the dsFab Dataset
dsFab.Tables.Add(dtPeople)
dsFab.Tables(0).TableName = "People"

' Add a row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Litwin"
drPeople("FirstName") = "Geoff"
drPeople("Email") = "geoff@theend.com"
dtPeople.Rows.Add(drPeople)

' Add another row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Elizabeth"
drPeople("FirstName") = "Anna"
drPeople("Email") = "anna@toddler.com"
```

Listing 15.5 continued

```
dtPeople.Rows.Add(drPeople)

' And a third row, this time using an array of values
' for the fields. Notice that you need to
' pass the AutoIncrement field a value of Nothing
Dim avalues() As Object = {Nothing, "Lea", "Suzanne", "s1@seattle.net"}
dtPeople.Rows.Add(avales)

Return dsFab.Tables("People").DefaultView
End Function
</script>
</head>
<body>
<form runat="server">
<asp:Label id="lblOut" runat="server"
Text="

## Fabricated DataSet Example

" />

<asp:DataGrid id="dgrPeople" runat="server"
BorderColor="black"
BorderWidth="1"
GridLines="Both"
CellPadding="3"
CellSpacing="0"
Font-Name="Verdana"
Font-Size="8pt"
HeaderStyle-BackColor="#99ccff"
AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>
```

The `FabricateDataSet` function in Listing 15.5 is responsible for creating the `dsFab` `DataSet` and populating its `People` `DataTable` with three rows of data.

Creating Columns

The code in `FabricateDataSet` creates `DataColumn`s using code like this (this particular snippet of code creates the `LastName` column):

```
' Create the LastName column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "LastName"
End With
dtPeople.Columns.Add(dc)
```

One of the columns in the `dtPeople` `DataTable`, `PersonId`, is made autoincrementing by setting its `AutoIncrement` property to `True`:

```
' Create the PersonId Column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.Int32")
    .ColumnName = "PersonId"
    .Unique = True
    .AutoIncrement = True
    .AutoIncrementSeed = 101
    .AutoIncrementStep = 1
End With
dtPeople.Columns.Add(dc)
```

Creating the PrimaryKey Constraint

Once the columns have been created, the `PrimaryKey` constraint for the `DataTable` is set using the following code:

```
Dim dcPk() As DataColumn = {dtPeople.Columns("PersonId")}
dtPeople.PrimaryKey = dcPk
```

Notice that the `dcPK` column is defined as an array. This is necessary even when the primary key contains only a single column, as in this example.

The following code is used to add the `dtPeople` `DataTable` to the `Tables` collection of the `dsFab` `DataSet`:

```
dsFab.Tables.Add(dtPeople)
dsFab.Tables(0).TableName = "People"
```

Adding Rows to a DataSet

Once created, you can add rows to the fabricated `DataSet` table in one of two ways:

1. Use the `NewRow` method of the `DataTable` to create a new empty row. Set the values of each of the columns of the row and append the new row to the `Rows` collection of the `DataTable` using the `Add` method.
2. Create an array of type `Object` and append the array to the `Rows` collection of the `DataTable` using the `Add` method.

This first technique is used for the inserting of the first and second rows. Here's the code that adds the first row to the `dtPeople` `DataTable`:

```
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Litwin"
drPeople("FirstName") = "Geoff"
drPeople("Email") = "geoff@theend.com"
dtPeople.Rows.Add(drPeople)
```

The second technique is used to insert the third row into dtPeople:

```
Dim avalues() As Object = {Nothing, "Lea", "Suzanne", "sl@seattle.net"}
dtPeople.Rows.Add(avales)
```

Notice that the value `Nothing` is used to skip the first `AutoIncrement` column when using the second technique.

The page resulting from `DataSetFab.aspx` is shown in Figure 15.4.



Figure 15.4

This `DataGrid` is bound to a fabricated `DataSet`.

Creating a DataSet from XML

So far in this chapter you've seen how to create a `DataSet` from database data and how to fabricate a `DataSet` from code. The last way to create a `DataSet` is to use XML.

You can use the `ReadXml` method of the `DataSet` object to read XML data and schema into a `DataSet`. The `DataSetFromXML1.aspx` page, shown in Listing 15.6 illustrates how to create a new `DataSet` from an XML file containing both data and schema.

Listing 15.6 DataSetFromXML1.aspx—Creating a DataSet from an XML File

```
<%@ Page Language="VB" Debug="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.XML" %>

<html>
<head>
<title>DataSetFromXML1.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub
```

Listing 15.6 continued

```

Sub BindDataSet()
    ' Create the DataSet and bind to it
    dgrPeople.DataSource = FabricateDataSet()
    dgrPeople.DataBind()
End Sub

Function FabricateDataSet() As DataView
    ' Create new empty DataSet
    Dim dsFab As DataSet = New DataSet
    dsFab.DataSetName = "PeopleSet"

    Call GrabXML(dsFab, Server.MapPath("morepeople.xml"))

    Return dsFab.Tables("People").DefaultView
End Function

Sub GrabXML(ByRef ds As DataSet, strFile As String)
    ' Create a FileStream object for input
    Dim fs As New System.IO.FileStream _
        (strFile, System.IO.FileMode.Open)
    ' Create an XmlTextReader to read the file
    Dim xtr As New System.Xml.XmlTextReader(fs)
    ' Read the XML document and schema into the DataSet
    ds.ReadXml(xtr, XmlReadMode.ReadSchema)
    ' Close the XmlTextReader
    xtr.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Creating a DataSet From XML

" />

    <asp:DataGrid id="dgrPeople" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#99ccff"
        AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>

```

The DataSetFromXML1.aspx page in Listing 15.6 should look familiar because it's similar to the DataSetFab.aspx page from Listing 15.5. The main difference is that schema and data are created from the morepeople.xml file. This magic is accomplished with the GrabXML subroutine shown here:

```
Sub GrabXML(ByRef ds As DataSet, strFile As String)
    ' Create a FileStream object for input
    Dim fs As New System.IO.FileStream _
        (strFile, System.IO.FileMode.Open)
    ' Create an XmlTextReader to read the file
    Dim xtr As New System.Xml.XmlTextReader(fs)
    ' Read the XML document and schema into the DataSet
    ds.ReadXml(xtr, XmlReadMode.ReadSchema)
    ' Close the XmlTextReader
    xtr.Close()
End Sub
```

This code opens a `FileStream` object in order to read the `morepeople.xml` file into memory. An `XmlTextReader` object is then employed to process the XML read into the `FileStream` object. Finally, the `DataSet`'s `ReadXml` method is used to read the XML into the `DataSet` and the `XmlTextReader` is then closed.

Notice the second parameter passed to the `ReadXml` method. This parameter is used to set the read mode employed by the `XmlTextReader`. This example uses a value of `XmlReadMode.ReadSchema` to force the `XmlTextReader` to read both the data and the schema. `XmlReadMode` can be set to any of the enumerated values shown in Table 15.1.

Table 15.1 `XmlReadMode` Enumeration Values

Value	Description
Auto	Automatically infers which <code>XmlReadMode</code> to use based on the XML. (Default.)
DiffGram	Reads an XML DiffGram, applying the changes to the <code>DataSet</code> .
Fragment	Reads XML documents containing inline XDR schema fragments, such as those created by SQL Server.
IgnoreSchema	Ignores any schema in the file and loads the data.
InferSchema	Ignores any schema and infers the schema from the data.
ReadSchema	Reads the schema and loads the data.

There may be times where you have a `DataSet` and its `DataTables` already created and you wish to append data from an XML file into the `DataSet`. `DataSetFromXML2.aspx` page found in Listing 15.7 does just this, ignoring the schema from the XML file (using `XmlReadMode.IgnoreSchema`) as it appends the XML rows to the existing `DataSet`.

Listing 15.7 `DataSetFromXML2.aspx`—This Page Adds Rows from the `Morepeople.xml` File to the Existing `DataSet`

```
<%@ Page Language="VB" Debug="true" %>
<%@ Import Namespace="System.Data" %>
```

Listing 15.7 continued

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

<html>
<head>
<title>DataSetFromXml12.aspx</title>
<script language="VB" runat="server">
Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub
Sub BindDataSet()
    ' Create the DataSet and bind to it
    dgrPeople.DataSource = FabricateDataSet()
    dgrPeople.DataBind()
End Sub
Function FabricateDataSet() As DataView
    ' Create new empty DataSet
    Dim dsFab As DataSet = New DataSet
    dsFab.DataSetName = "PeopleSet"
    Dim dtPeople As DataTable = New DataTable
    Dim dc As DataColumn
    Dim drPeople As DataRow

    ' Create the PersonId Column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.Int32")
        .ColumnName = "PersonId"
        .Unique = True
        .AutoIncrement = True
        .AutoIncrementSeed = 101
        .AutoIncrementStep = 1
    End With
    dtPeople.Columns.Add(dc)

    ' Create the LastName column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.String")
        .ColumnName = "LastName"
    End With
    dtPeople.Columns.Add(dc)

    ' Create the FirstName column
    dc = New DataColumn
```

Listing 15.7 continued

```
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "FirstName"
End With
dtPeople.Columns.Add(dc)

' Create the EMail column
dc = New DataColumn
With dc
    .DataType = System.Type.GetType("System.String")
    .ColumnName = "Email"
End With
dtPeople.Columns.Add(dc)

' Make PersonId the primary key of the DataTable
Dim dcPK() As DataColumn = {dtPeople.Columns("PersonId")}
dtPeople.PrimaryKey = dcPK

' Add the dtPeople DataTable to the dsFab Dataset
dsFab.Tables.Add(dtPeople)
dsFab.Tables(0).TableName = "People"

' Add a row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Litwin"
drPeople("FirstName") = "Geoff"
drPeople("Email") = "geoff@theend.com"
dtPeople.Rows.Add(drPeople)

' Add another row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Elizabeth"
drPeople("FirstName") = "Anna"
drPeople("Email") = "anna@toddler.com"
dtPeople.Rows.Add(drPeople)

' And a third row, this time using an array of values
' for the fields. Notice that you need to
' pass the AutoIncrement field a value of Nothing
Dim avalues() As Object = {Nothing, "Lea", "Suzanne", "sl@seattle.net"}
dtPeople.Rows.Add(avvalues)

Call GrabXML(dsFab, Server.MapPath("morepeople.xml"))

Return dsFab.Tables("People").DefaultView
End Function

Sub GrabXML(ByRef ds As DataSet, strFile As String)
    ' Create a FileStream object for input
```

Listing 15.7 continued

```
Dim fsReadXml As New System.IO.FileStream _
(strFile, System.IO.FileMode.Open)
' Create an XmlTextReader to read the file
Dim xr As New System.Xml.XmlTextReader(fsReadXml)
' Read the XML document into the DataSet, ignoring
' the schema since the DataTable is already created
ds.ReadXml(xr, XmlReadMode.IgnoreSchema)
' Close the XmlTextReader
xr.Close()
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Appending XML Rows to a DataSet

" />

    <asp:DataGrid id="dgrPeople" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#99ccff"
        AlternatingItemStyle-BackColor="lightgray" />
</form>
</body>
</html>
```

The DataSetFromXML2.aspx page in Listing 15.7 is shown in Figure 15.5.

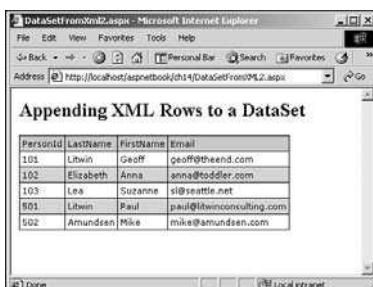


Figure 15.5

Adding rows from an XML file to an existing fabricated DataSet.

TIP

You can use the ReadXmlSchema method to read an XML schema into a DataSet without grabbing any of the data.

Generating XML from a DataSet

The flip side of the creation of a DataSet from XML is the creation of XML from a DataSet. The DataSet object supports several methods for generating XML from a DataSet. These methods are summarized in Table 15.2.

Table 15.2 Methods of the DataSet Object for Generating XML

Method	Description
GetXml	Returns a string containing the XML data representation of the DataSet.
GetXmlSchema	Returns a string containing the XML schema representation of the DataSet.
WriteXml	Outputs the XML representation of the data (and optionally its schema) into various objects, including Stream, StreamWriter, and XmlWriter objects.
WriteXmlSchema	Outputs the XML schema representation of the DataSet into various objects, including Stream, StreamWriter, and XmlWriter objects.

The DataSetToXML.aspx page, shown in Listing 15.8 illustrates the use of the WriteXml method to create an XML file from a DataSet. When the button control is clicked, code attached to the button's Click event uses the WriteXml method to output the XML representation of the DataSet to the people.xml file.

Listing 15.8 DataSetToXML.aspx—Outputting the XML Representation of the DataSet to the People.xml File

```
<%@ Page Language="VB" Debug="true" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.XML" %>

<html>
<head>
<title>DataSetToXml.aspx</title>
<script language="VB" runat="server">

Sub Page_Load(Src As Object, E As EventArgs)
    ' Don't need to rebind for postbacks
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If
End Sub
```

Listing 15.8 continued

```
Sub BindDataSet()
    ' Create the DataSet and bind to it
    dgrPeople.DataSource = FabricateDataSet()
    dgrPeople.DataBind()
End Sub

Function FabricateDataSet() As DataView
    ' Create new empty DataSet
    Dim dsFab As DataSet = New DataSet
    dsFab.DataSetName = "PeopleSet"
    Dim dtPeople As DataTable = New DataTable
    Dim dc As DataColumn
    Dim drPeople As DataRow

    ' Create the PersonId Column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.Int32")
        .ColumnName = "PersonId"
        .Unique = True
        .AutoIncrement = True
        .AutoIncrementSeed = 101
        .AutoIncrementStep = 1
    End With
    dtPeople.Columns.Add(dc)

    ' Create the LastName column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.String")
        .ColumnName = "LastName"
    End With
    dtPeople.Columns.Add(dc)

    ' Create the FirstName column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.String")
        .ColumnName = "FirstName"
    End With
    dtPeople.Columns.Add(dc)

    ' Create the EMail column
    dc = New DataColumn
    With dc
        .DataType = System.Type.GetType("System.String")
        .ColumnName = "Email"
    End With
End Function
```

Listing 15.8 continued

```
End With
dtPeople.Columns.Add(dc)

' Make PersonId the primary key of the DataTable
Dim dcPK() As DataColumn = {dtPeople.Columns("PersonId")}
dtPeople.PrimaryKey = dcPK

' Add the dtPeople DataTable to the dsFab Dataset
dsFab.Tables.Add(dtPeople)
dsFab.Tables(0).TableName = "People"

' Add a row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Litwin"
drPeople("FirstName") = "Geoff"
drPeople("Email") = "geoff@theend.com"
dtPeople.Rows.Add(drPeople)

' Add another row to dtPeople
drPeople = dtPeople.NewRow()
drPeople("LastName") = "Elizabeth"
drPeople("FirstName") = "Anna"
drPeople("Email") = "anna@toddler.com"
dtPeople.Rows.Add(drPeople)

' And a third row, this time using an array of values
' for the fields. Notice that you need to
' pass the AutoIncrement field a value of Nothing
Dim avalues() As Object = {Nothing, "Lea", "Suzanne", "sl@seattle.net"}
dtPeople.Rows.Add(aValues)

' Cache the DataSet so we can use it when
' the button is clicked
Cache("dsFab") = dsFab

Return dsFab.Tables("People").DefaultView
End Function

Sub OutputXML(ByRef ds As DataSet, strFile As String)
    Dim fs As New System.IO.FileStream _
        (strFile, System.IO FileMode.Create)
    ' Create an XmlTextWriter with the fileStream.
    Dim xtw As New System.Xml.XmlTextWriter _
        (fs, System.Text.Encoding.Unicode)

    ' Write to the file with the WriteXml method.
    ds.WriteXml(xtw, XmlWriteMode.WriteSchema)
```

Listing 15.8 continued

```

    xtw.Close()
End Sub

Sub GenerateXML(Src As Object, E As EventArgs)
    Call OutputXML(Cache("dsFab"), Server.MapPath("people.xml"))
    lblMsg.Text = "XML written to " & Server.MapPath("people.xml")
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Fabricated DataSet Example

" />

    <asp:DataGrid id="dgrPeople" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#99ccff"
        AlternatingItemStyle-BackColor="lightgray" />
    <p><asp:Button id="cmdWrite" runat="server"
        Text="GenerateXML" OnClick="GenerateXML" /></p>
    <asp:Label id="lblMsg" runat="server"
        style="Color:red" />
</form>
</body>
</html>

```

The code from `DataSetToXML.aspx` that outputs the `DataSet` to an XML file is contained in the `OutputXML` subroutine shown here:

```

Sub OutputXML(ByRef ds As DataSet, strFile As String)
    Dim fs As New System.IO.FileStream _
        (strFile, System.IO FileMode.Create)
    ' Create an XmlTextWriter with the fileStream.
    Dim xtw As New System.Xml.XmlTextWriter _
        (fs, System.Text.Encoding.Unicode)

    ' Write to the file with the WriteXml method.
    ds.WriteXml(xtw, XmlWriteMode.WriteSchema)
    xtw.Close()
End Sub

```

The second parameter passed to the `WriteXml` method is used to control the type of output generated by the `WriteXml` method. The code in Listing 15.8 uses a value of `XmlWriteMode.WriteSchema` to force the writing of both data and schema. `XmlWriteMode` can be set to any of the enumerated values shown in Table 15.3.

Table 15.3 `XmlWriteMode` Enumeration Values

Value	Description
DiffGram	Writes the <code>DataSet</code> as a DiffGram.
IgnoreSchema	Writes the data but no schema.
WriteSchema	Writes both the data and schema. (Default.)

The `DataSetToXML.aspx` page is shown in Figure 15.6.

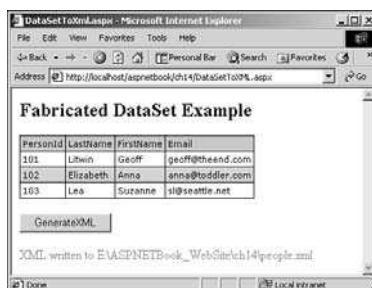


Figure 15.6

Moving the data and schema of a `DataSet` into an XML file.

Updating DataSet Data

As you saw earlier in this chapter in the “Fabricating `DataSets`” section, the `DataSet` class supports the insertion of rows. As you might expect, you can also update and delete rows from a `Dataset`. Furthermore, using a `SqlDataAdapter` or `OleDbDataAdapter` object, you use the updated rows of the `DataSet` to update the original data source.

To update a row in a `DataTable`, simply navigate to the row you wish to update and set the fields in the `DataRow` to the new values like this:

```
datarow("field1") = newvalue1  
datarow("field2") = newvalue2  
...  
datarow("fieldx") = newvaluex
```

To delete a row in a `DataTable`, navigate to the row and execute the `Delete` method on the `DataRow` like this:

```
datarow.Delete
```

Managing the Changed Status of the DataSet

DataSets maintain a record of all changes that have been made to their data. To determine if a DataSet contains any modified, deleted, or inserted rows you can use the HasChanges method which returns True if the DataSet contains changes or False if it has not been modified since it was loaded or since an AcceptChanges or RejectChanges method was last executed. Here's the basic syntax:

```
booleanvariable = dataset.HasChanges()
```

You can also pass the HasChanges method one of the DataRowState enumerated values to check if the DataSet contains a certain type of change. The DataRowState values, which are also returned by the RowState property of individual DataRows are summarized in Table 15.4.

Table 15.4 *DataRowState Enumeration Values*

Value	Description
Added	Inserted row.
Deleted	Deleted row.
Detached	Row that has been created but not added to the Rows collection.
Modified	Modified row.
Unchanged	Unmodified row.

For example, if you wished to check if the DataSet contained any deleted rows, you could use code like this:

```
booleanvariable = dataset.HasChanges(DataRowState.Deleted)
```

You can use the GetChanges method to create a new DataSet containing all of the changed rows in a DataSet. Here's the syntax:

```
datasetofchanges = dataset.GetChanges()
```

As was the case with the HasChanges method, you can pass the GetChanges method a DataRowState value to create a new DataSet containing only certain types of changed rows.

You can rollback all of the changes made to a DataSet by calling the RejectChanges method

```
dataset.RejectChanges()
```

You can accept all of the changes made to a DataSet, resetting the HasChanges state of the DataSet and all of its DataTables by calling the AcceptChanges method:

```
dataset.AcceptChanges()
```

TIP

You can also call RejectChanges and AcceptChanges on individual DataTable and DataRow objects.

Writing Updates Back to the Database

Being able to make changes to rows in a `DataSet` wouldn't be that useful unless you couldn't somehow synchronize the changes with the original data source from where the rows came. This is made possible by the `Update` methods of the `SqlDataAdapter` and `OleDbDataAdapter` classes.

Before you can use the `Update` method, you must configure the appropriate `DataAdapter`, telling it how to update the data source: either by using SQL statements (or stored procedures) that you supply, or by using SQL statements that are generated by the appropriate `CommandBuilder` class.

Using CommandBuilder Generated SQL Statements

You can use a `SqlCommandBuilder` or `OleDbCommandBuilder` object to generate one or more SQL statements that the appropriate `DataAdapter` object can use to update the original data source with the changed `DataSet` records.

Here are the basic steps:

1. Create a `SqlConnection/OleDbConnection` to the database.
2. Create a `SqlDataAdapter/OleDbDataAdapter` object, passing it a SQL SELECT statement and the `SqlConnection/OleDbConnection` object for Step 1.
3. Create a `SqlCommandBuilder/OleDbCommandBuilder` object, passing it the `SqlDataAdapter/OleDbDataAdapter` object from Step 2.
- 4a. If you will be handling updated rows, set the `UpdateCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to the `GetUpdateCommand` method of the `SqlCommandBuilder/OleDbCommandBuilder` object.
- 4b. If you will be handling deleted rows, set the `DeleteCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to the `GetDeleteCommand` method of the `SqlCommandBuilder/OleDbCommandBuilder` object.
- 4c. If you will be handling inserted rows, set the `InsertCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to the `GetInsertCommand` method of the `SqlCommandBuilder/OleDbCommandBuilder` object.
5. Execute the `Update` method of the `SqlDataAdapter/OleDbDataAdapter` object, passing it the `DataSet` object and the name of the `DataTable` to update. The `Update` method returns an `Integer` value containing the number of rows that were updated in the data source.
6. Call the `AcceptChanges` method of the `DataSet` object to reset the `RowState` property of all the changed rows in the `DataSet` to `Unchanged`.

The code in Listing 15.9, from `SqlDataSetUpdate.aspx`, shows the process of synchronizing updated and deleted records from the `Employees` `DataTable` of the `ds` `DataSet` to the `Employees` table of the SQL Server Northwind database.

Listing 15.9 SqlDataSetUpdate.aspx—Passing Updated and Deleted Rows from the Employees DataTable of the ds DataSet to the Northwind Employees Table

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

Listing 15.9 continued

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

<html>
<head>
<title>SqlDataSetUpdate.aspx</title>
<script language="VB" runat="server">
' ds variable is global to page
Dim ds as DataSet

Sub Page_Load(Src As Object, E As EventArgs)
    Call GetDataSet()
    ' Only need to bind the first time through
    ' Otherwise, binding will occur after the
    ' Edit/Update/Cancel events.
    If Not Page.IsPostBack Then
        Call BindDataSet()
    End If

    ' Enable Commit/Abandon buttons if DataSet
    ' has pending changes.
    Call EnableDbButtons(ds.HasChanges())
End Sub

Sub BindDataSet()
    ' Bind ds DataSet to DataGrid

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

Sub GetDataSet()
    ' Check if dataset has already been built and
    ' stored in Session. If not, then build it.

    If Session("ds") Is Nothing Then
        ' Build DataSet from database
        Call BuildDataSet()
    Else
        ' Retrieve DataSet from session variable
        ds = Session("ds")
    End If
End Sub

Sub BuildDataSet()
    ' This routine builds the DataSet from
    ' the Northwind Employees table.

    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
```

Listing 15.9 continued

```
Dim strSQL As String = "SELECT * FROM Employees"

' Create connection to SQL Server Northwind database
Dim cnx As SqlConnection = New SqlConnection(strCnx)
' Create & execute SqlDataAdapter query to return records
Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)

' Make sure we can update rows by forcing the inclusion
' of the primary key columns. Not really necessary in this example.
sda.MissingSchemaAction = MissingSchemaAction.AddWithKey

' Create new empty DataSet
ds = New DataSet

' Fill DataSet using SqlDataAdapter
sda.Fill(ds, "Employees")

' Set the PrimaryKey for Employees DataTable
Dim dcPK() As DataColumn = {ds.Tables("Employees").Columns("EmployeeId")}
ds.Tables("Employees").PrimaryKey = dcPK

' Store DataSet to session variable
Session("ds") = ds

cnx.Close()
End Sub

Sub EnableDbButtons(blnEnable)
    ' Helper routine to enable/disable
    ' Database Update buttons.

    cmdDbCommit.Enabled = blnEnable
    cmdDbAbandon.Enabled = blnEnable

    ' If enabled then DataSet must be dirty.
    If blnEnable Then
        lblMsg.Text = "Pending changes."
    End If
End Sub

Sub GridEdit(sender As Object, e As DataGridViewEventArgs)
    ' This code is executed when the Edit button
    ' associated with the EditCommandColumn is clicked.

    ' Set the EditItemIndex to the index of the row
    ' that triggered the Edit event
```

Listing 15.9 continued

```
dgrEmployees.EditItemIndex = e.Item.ItemIndex
Call BindDataSet()

' Disable buttons and clear message during editing
Call EnableDbButtons(False)
lblMsg.Text = ""
End Sub

Sub GridCancel(sender As Object, e As DataGridCommandEventArgs)
    ' This code is executed when the Cancel button
    ' associated with the EditCommandColumn is clicked.

    ' Reset EditItemIndex to -1 which takes us out of edit mode
    dgrEmployees.EditItemIndex = -1
    Call BindDataSet()
End Sub

Sub GridUpdate(sender As Object, e As DataGridCommandEventArgs)
    ' This code is executed when the Update button
    ' associated with the EditCommandColumn is clicked.

    Dim drFound As DataRow

    ' Find row in DataSet
    drFound = ds.Tables("Employees").Rows.Find(e.Item.Cells(2).Text)
    If Not drFound Is Nothing Then
        ' Grab controls from DataGrid
        Dim ctlFirstName As TextBox = e.Item.Cells(3).Controls(0)
        Dim ctlLastName As TextBox = e.Item.Cells(4).Controls(0)
        Dim ctlPhone As TextBox = e.Item.Cells(5).Controls(0)

        ' Set DataRow values to DataGrid control values
        drFound("FirstName") = ctlFirstName.Text
        drFound("LastName") = ctlLastName.Text
        drFound("HomePhone") = ctlPhone.Text
    Else
        lblMsg.Text = "Error: Row not found"
    End if

    ' Reset EditItemIndex to -1 which takes us out of edit mode
    dgrEmployees.EditItemIndex = -1
    Session("ds") = ds
    Call BindDataSet()

    ' Need to fix up buttons.
    Call EnableDbButtons(ds.HasChanges())
End Sub
```

Listing 15.9 continued

```
Sub GridCommand(sender As Object, e As DataGridCommandEventEventArgs)
    ' This code is executed when a Command button is clicked.
    ' The only Command button defined is for deleting rows.

    If e.CommandName = "Delete" Then
        Dim drFound As DataRow

        ' Find row in DataSet
        drFound = ds.Tables("Employees").Rows.Find(e.Item.Cells(2).Text)
        If Not drFound Is Nothing Then
            ' Grab controls from DataGrid
            drFound.Delete
        Else
            lblMsg.Text = "Error: Row not found"
        End if

        ' Reset EditItemIndex to -1 which takes us out of edit mode
        dgrEmployees.EditItemIndex = -1
        Session("ds") = ds
        Call BindDataSet()

        ' Need to fix up buttons.
        Call EnableDbButtons(ds.HasChanges())
    Else
        ' Shouldn't happen
    End If
End Sub

Sub DbCommit(Src As Object, E As EventArgs)
    ' This routine is called when the 'Commit Changes'
    ' button is clicked. It sends all pending changes
    ' to the database.

    Dim intRows As Integer
    Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
    Dim strSQL As String = "SELECT * FROM Employees"

    ' Create connection to SQL Server Northwind database
    Dim cnx As SqlConnection = New SqlConnection(strCnx)
    ' Create & execute SqlDataAdapter query to return records
    Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)

    ' Create SqlCommandBuilder to handle Update statement
    Dim scb As SqlCommandBuilder = New SqlCommandBuilder(sda)
    sda.UpdateCommand = scb.GetUpdateCommand()
    sda.DeleteCommand = scb.GetDeleteCommand()
```

Listing 15.9 continued

```
' Call Update method on Employees DataTable
intRows = sda.Update(ds, "Employees")
' Reset state of edited rows
ds.AcceptChanges()

Call BindDataSet()
cnx.Close()
lblMsg.Text = intRows & " database rows updated."

' Need to fix up buttons.
Call EnableDbButtons(ds.HasChanges())
End Sub

Sub DbAbandon(Src As Object, E As EventArgs)
    ' This routine is called when the 'Abandon Changes'
    ' button is clicked. It forces a rebuild of the
    ' DataSet from the database.

    ' Rebuild DataSet from database
    Call BuildDataSet()
    Call BindDataSet()
    lblMsg.Text = "All changes abandoned.

    ' Need to fix up buttons.
    Call EnableDbButtons(ds.HasChanges())
End Sub
</script>
</head>
<body>
<form runat="server">
    <asp:Label id="lblOut" runat="server"
        Text="

## Northwind Employees

" />

    <asp:DataGrid id="dgrEmployees" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#99ccff"
        AlternatingItemStyle-BackColor="lightgray"
        AutoGenerateColumns="False"
        OnEditCommand="GridEdit"
        OnCancelCommand="GridCancel"
        OnUpdateCommand="GridUpdate"
```

Listing 15.9 continued

```
OnItemCommand="GridCommand"
>
<Columns>
    <asp:EditCommandColumn
        EditText="Edit"
        UpdateText="Save"
        CancelText="Cancel"
        ItemStyle-Wrap="false"
        HeaderText="Edit"
        HeaderStyle-Wrap="false" />
    <asp:ButtonColumn
        HeaderText="Delete"
        Text="Delete"
        CommandName="Delete" />
    <asp:BoundColumn HeaderText="Employee Id" DataField="EmployeeId"
        ReadOnly=True/>
    <asp:BoundColumn HeaderText="First Name" DataField="FirstName" />
    <asp:BoundColumn HeaderText="Last Name" DataField="LastName" />
    <asp:BoundColumn HeaderText="Phone" DataField="HomePhone" />
</Columns>
</asp:DataGrid>
<br />
<asp:Button id="cmdDbCommit" runat="server"
    Text="Commit Changes" OnClick="DbCommit"/>
<asp:Button id="cmdDbAbandon" runat="server"
    Text="Abandon Changes" OnClick="DbAbandon"/>
<br /><br />
<asp:Label id="lblMsg" runat="server"
    Style="Color:red"/>
</form>
</body>
</html>
```

The `SqlDataSetUpdate.aspx` example includes quite a bit of code that deals with making edits to a `DataGrid`. The `DataGrid` code should look similar to code you saw earlier in this book in Chapter 10, “Designing Advanced Interfaces with Web Form List Controls.”

The code that updates the database with the updated and deleted records from the `DataSet` can be found in the `DbCommit` subroutine. The first part of the subroutine constructs the `SqlConnection` and `SqlDataAdapter` objects:

```
Sub DbCommit(Src As Object, E As EventArgs)
    ' This routine is called when the 'Commit Changes'
    ' button is clicked. It sends all pending changes
    ' to the database.
```

```

Dim intRows As Integer
Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
Dim strSQL As String = "SELECT * FROM Employees"

' Create connection to SQL Server Northwind database
Dim cnx As SqlConnection = New SqlConnection(strCnx)
' Create & execute SqlDataAdapter query to return records
Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)

```

Next, the `DbCommit` code creates the `SqlCommandBuilder` object and uses it to construct the `UpdateCommand` and `DeleteCommand` properties of the `SqldataAdapter`:

```

' Create SqlCommandBuilder to handle Update statement
Dim scb As SqlCommandBuilder = New SqlCommandBuilder(sda)
sda.UpdateCommand = scb.GetUpdateCommand()
sda.DeleteCommand = scb.GetDeleteCommand()

```

The code then calls the `Update` method and resets the status of the rows in the `DataSet` using the `AcceptChanges` method:

```

' Call Update method on Employees DataTable
intRows = sda.Update(ds, "Employees")
' Reset state of edited rows
ds.AcceptChanges()

```

The rest of the code in `DbCommit` is housekeeping and cleanup code to make sure the `DataGridView` and database update buttons are synchronized with the state of the `DataSet`. The code also informs the user as to how many rows were updated:

```

Call BindDataSet()
cnx.Close()
lblMsg.Text = intRows & " database rows updated."

' Need to fix up buttons.
Call EnableDbButtons(ds.HasChanges())
End Sub

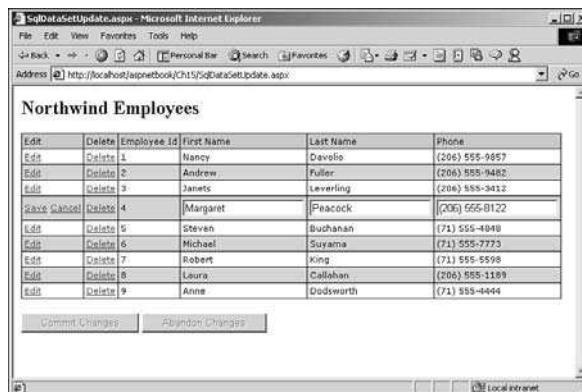
```

One other detail worth mentioning: If the SQL statements that are used to create the `DataTables` in the `DataSet` do not include the primary key columns from the original data source, you will need to set the `MissingSchemaAction` property of the `SqldataAdapter/OleDbDataAdapter` object when creating the `DataSet` to `MissingSchemaAction.AddWithKey` as shown in this code from the `BuildDataSet` subroutine in Listing 15.9.

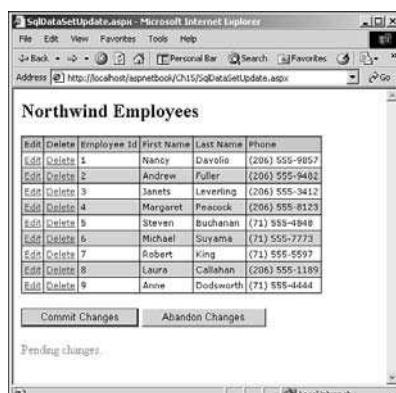
```
sda.MissingSchemaAction = MissingSchemaAction.AddWithKey
```

In the case of Listing 15.9, this statement was unnecessary because the SQL statement requested all of the columns from the Employees table.

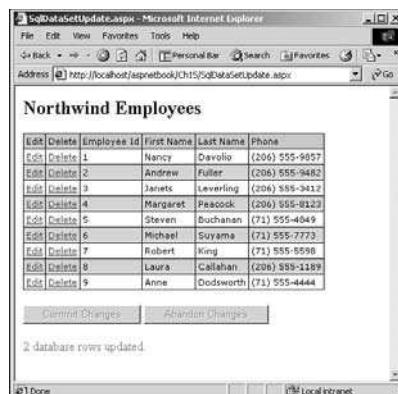
The SqlDataSetUpdate.aspx page is shown in action in Figures 15.7, 15.8, and 15.9.

**Figure 15.7**

Editing a row of data in place using the DataGrid control.

**Figure 15.8**

The Commit Changes and Abandon Changes buttons are enabled because uncommitted rows are present in the DataSet.

**Figure 15.9**

Two rows have updated in the Northwind Employees table from edits made to the DataSet by clicking the Commit Changes button.

Supplying Custom SQL Statements

You don't have to use the SQL statements generated by the `CommandBuilder` object. Instead, you can supply the appropriate `DataAdapter` object with your own custom SQL statements or stored procedures that you wish it to use when updating the database.

Here are the basic steps involved in updating the database using custom SQL statements:

1. Create a `SqlConnection/OleDbConnection` to the database.
2. Create a `SqlDataAdapter/OleDbDataAdapter` object, passing it a SQL SELECT statement and the `SqlConnection/OleDbConnection` object for step 1.
- 3a. If you will be handling updated rows, set the `UpdateCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to a `SqlCommand/OleDbCommand` object that points to the stored procedure that will be making the updates.
- 3b. If you will be handling deleted rows, set the `DeleteCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to a `SqlCommand/OleDbCommand` object that points to the stored procedure that will be making the deletions.
- 3c. If you will be handling inserted rows, set the `InsertCommand` property of the `SqlDataAdapter/OleDbDataAdapter` object to a `SqlCommand/OleDbCommand` object that points to the stored procedure that will be making the inserts.
4. Create `SqlParameter/OleDbParameter` objects for each of the parameters used in steps 3a-3c. Make sure to specify the `SourceVersion`, and `SourceColumn` properties of each parameter.

5. Execute the Update method of the SqlDataAdapter/OleDbDataAdapter object, passing it the DataSet object and the name of the DataTable to update. The Update method returns an Integer value containing the number of rows that were updated in the data source.
6. Call the AcceptChanges method of the DataSet object to reset the RowState property of all the changed rows in the DataSet to Unchanged.

Listing 15.10, taken from SqlDataSetUpdateSP.aspx, illustrates how to use a stored procedure to synchronize updates to the DataSet with the original data source, the Employees table in the SQL Server Northwind database. It passes updated rows from the Employees DataTable of the ds DataSet to the Northwind Employees table, employing the procUpdateEmployee1 stored procedure (created by the CreateSP subroutine) to update the rows. The remainder of the code in SqlDataSetUpdateSP.aspx is very similar to the code in Listing 15.9.

Listing 15.10 SqlDataSetUpdateSP.aspx—Synchronizing Updates to the DataSet

```
Sub CreateSP(cnxOpen)
    ' This routine creates the procUpdateEmployee1
    ' stored procedure in the Northwind database.
    ' It uses Try/Catch to avoid the error where
    ' the stored procedure is already defined.

    Dim strSQL As String =
        "CREATE PROCEDURE procUpdateEmployee1 " & _
        "@employeeid INTEGER, " & _
        "@lastname VARCHAR(20), " & _
        "@firstname VARCHAR(20), " & _
        "@homephone VARCHAR(20) " & _
        "AS " & _
        "UPDATE Employees " & _
        "SET LastName = @lastname, " & _
        "    FirstName = @firstname, " & _
        "    HomePhone = @homephone " & _
        "WHERE EmployeeId = @employeeid"

    ' Execute SqlCommand query to create stored proc
    Try
        Dim cmdCreateSP As SqlCommand = New SqlCommand(strSQL, cnxOpen)
        cmdCreateSP.ExecuteNonQuery()
    Catch
        ' Ignore error that will occur if stored procedure
        ' is already defined.
    End Try
End Sub

Sub DbCommit(Src As Object, E As EventArgs)
```

Listing 15.10 continued

```
' This routine is called when the 'Commit Changes'
' button is clicked. It sends all pending changes
' to the database.

Dim prm As SqlParameter
Dim intRows As Integer
Dim strCnx As String = "server=localhost;uid=sa;pwd=;database=northwind;"
Dim strSQL As String = "SELECT * FROM Employees"

' Create connection to SQL Server Northwind database.
Dim cnx As SqlConnection = New SqlConnection(strCnx)
cnx.Open()

' Create & execute SqlDataAdapter query to return records.
Dim sda As SqlDataAdapter = New SqlDataAdapter(strSQL, cnx)

' Create stored procedure if necessary
Call CreateSP(cnx)

' Create SqlCommand to handle Update statement using stored procedure.
Dim cmdUpdate As SqlCommand = New SqlCommand("procUpdateEmployee1", cnx)
cmdUpdate.CommandType = CommandType.StoredProcedure

' Create first parameter.
prm = New SqlParameter("@employeeid", SqlDbType.Int)
prm.Direction = ParameterDirection.Input
prm.SourceColumn = "EmployeeId"
prm.SourceVersion = DataRowVersion.Original
cmdUpdate.Parameters.Add(prm)

' Create second parameter.
prm = New SqlParameter("@lastname", SqlDbType.VarChar, 20)
prm.Direction = ParameterDirection.Input
prm.SourceColumn = "LastName"
prm.SourceVersion = DataRowVersion.Current
cmdUpdate.Parameters.Add(prm)

' Create third parameter.
prm = New SqlParameter("@firstname", SqlDbType.VarChar, 20)
prm.Direction = ParameterDirection.Input
prm.SourceColumn = "FirstName"
prm.SourceVersion = DataRowVersion.Current
cmdUpdate.Parameters.Add(prm)

' Create fourth parameter.
prm = New SqlParameter("@homephone", SqlDbType.VarChar, 20)
prm.Direction = ParameterDirection.Input
```

Listing 15.10 continued

```
prm.SourceColumn = "HomePhone"
prm.SourceVersion = DataRowVersion.Current
cmdUpdate.Parameters.Add(prm)

' Set the UpdateCommand property of the SqlDataAdapter to
' point to SqlCommand object containing the stored procedure.
sda.UpdateCommand = cmdUpdate

' Call Update method on Employees DataTable.
intRows = sda.Update(ds, "Employees")
' Reset state of edited rows.
ds.AcceptChanges()

Call BindDataSet()
cnx.Close()
lblMsg.Text = intRows & " database rows updated."

' Need to fix up buttons.
Call EnableDbButtons(ds.HasChanges())
End Sub
```

Much of the parameter definition code shown in Listing 15.10 is identical to the code you saw in Chapter 14, in the section entitled “Working with Stored Procedure Parameters.” However, there are a couple of additional SqlParameter/OleDbParameter object properties not discussed in Chapter 14 that need to be set when using the SqlCommand/OleDbCommand objects to update data from a DataSet: SourceColumn and SourceVersion.

You set the SqlParameter/OleDbParameter object’s SourceColumn property to the name of the column in the DataSet that corresponds to the stored procedure parameter. For example:

```
prm.SourceColumn = "EmployeeId"
```

You set the SqlParameter/OleDbParameter object’s SourceVersion property to a value indicating which version of the column in the DataSet you wish to use. You need to set SourceVersion to one of the following DataRowVersion enumeration values: Current or Original.

You can see in Listing 15.10 that the SourceVersion property is set to Original for the EmployeeId column and Current for the remaining columns. Original is used for the EmployeeId column because EmployeeId is used in the WHERE clause of the stored procedure to match up records by primary key value. (Actually, for this particular example either Current or Original could have been used because EmployeeId is a read-only field in the DataGrid.) For the other columns, Current is used to pass the current value—which may have been changed by the user—to the stored procedure.



NOTE

There are additional features of the `DataSet` that haven't been discussed in this chapter. These features include the ability to work with the collection of errors triggered by updates to the data and foreign-key constraints. See the .NET Framework SDK documentation for more details on these features.

Summary

In this chapter, you learned how to create and manipulate the ADO.NET `DataSet` from ASP.NET pages. You now know how to create `DataSets` from database tables, from code, and from XML files. You worked with a number of the `DataSet` objects, including `DataTables`, `DataViews`, and `DataRelations`.

You also learned how to update the data in a `DataSet` and pass back the changes to the original data source using the `SqlDataAdapter` and `OleDbDataAdapter` objects.

PART V

CREATING AND USING WEB SERVICES

- 16 Understanding the Web Service Model
- 17 Publishing Web Services with ASP.NET
- 18 Consuming Web Services with ASP.NET

CHAPTER 16

Understanding the Web Service Model

It could be argued that the most significant feature of the new ASP.NET programming environment is *Web Services*. Web Services is the name Microsoft has given a set of classes and technologies that allow programmers to publish executable methods via the Internet. This means that you can now create familiar-looking classes with method and argument and then allow other Web servers to call directly into these classes without requesting typical HTML pages. In essence, this provides a kind of COM communication for the Internet.

Even though this COM-style programming of Internet objects seems familiar to programmers experienced with Microsoft tools, Web Services is not based on Microsoft's proprietary COM protocol. Instead, it is based on an open standard called Simple Object Access Protocol (SOAP). This means that other platforms that understand the open protocol, such as Apache servers, can also call into the components the same way Microsoft Web servers can.

In this chapter, you learn just what Web Services are, how Web Services relate to the SOAP standard, and how you can design Web Services components to publish and consume. Finally, you'll also learn to deal with other important Web Services issues such as security, state management, and transactions.

Some Web Service History

One of the earliest published descriptions of an Internet protocol that would eventually become the basis of the SOAP protocol is known as XML-RPC or XML remote procedure calling. David Winer of Userland.com described XML-RPC

around 1998. Winer described the basics of how XML could be used to describe type library information in an operating-system-neutral way that could make it possible for then-incompatible component technologies to ‘talk’ to each other over the Internet.

Not long after Winer published his XML-RPC specifications, several other companies became interested in the project. Eventually Microsoft, IBM, Userland.com and other companies proposed a new protocol to the W3C organization in 1999. This protocol standard eventually became known as the Simple Object Access Protocol, or SOAP.

Since that time, the SOAP protocol has undergone a number of changes. At present, SOAP 1.1 is the current version supported by the W3C. Most major software vendors—including Microsoft, IBM, Sun, and others—have agreed upon this version. This means that solutions based on the SOAP 1.1 standard are likely to work in the most common scenarios encountered by Web programmers today.

NOTE

Although at this writing SOAP 1.1 is only at the NOTE status at W3C (the lowest level of recommendation at W3C), it is assumed that the present version will eventually become an officially recommended protocol with only minor changes.

The SOAP Protocol

In its simplest form, the SOAP protocol is an XML-based message format used to pass information between two locations or “endpoints.” This information is passed in what is known as an “envelope.” In Web Services, this envelope contains detailed information needed to properly marshal data types to and from a remote component.

Basically, Web Services uses the SOAP envelope to send type library information between two locations. Once this information is shared and understood, Web Services uses SOAP envelopes to send data into the remote component, and the remote component uses SOAP envelopes to return the resulting values back to the caller (see Figure 16.1).

The Web Service Description Language

In order for two machines at different locations, possibly using different component specifications or even different operating systems, to successfully communicate, they need to understand what methods are available and what input and output parameters are involved in the method calls. To handle this important step, Web Services uses the Web Service Description Language (WSDL). The WSDL message contains all the information needed to properly compose SOAP requests and responses.

Listing 16.1 shows the beginning portion of a WSDL message that describes a component that publishes a single method called `DoMath` that accepts two inputs and returns a single value as an output.

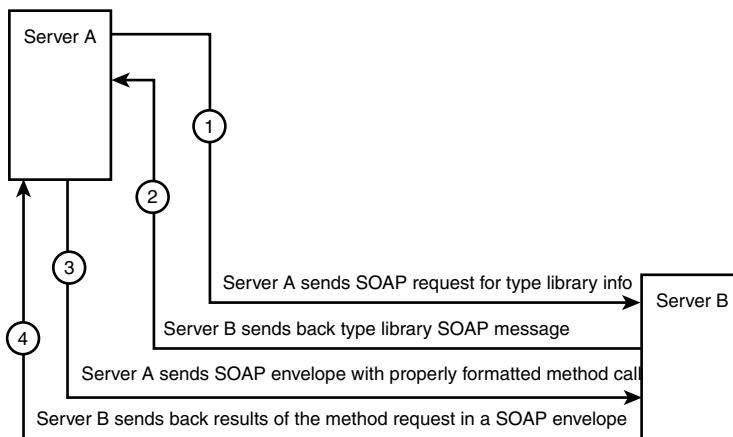
**Figure 16.1**

Illustration of a typical SOAP-based procedure call between two machines.

Listing 16.1 SAMPLE.WSDL—An Example WSDL Message for SOAP Services

```

<?xml version="1.0"?>
<definitions xmlns:s="http://www.w3.org/2000/10/XMLSchema" xmlns:http=
"http://schemas.xmlsoap.org/wsdl/http/" xmlns:mime=
"http://schemas.xmlsoap.org/wsdl/mime/" xmlns:urt=
"http://microsoft.com/urt/wsdl/text/" xmlns:soap=
"http://schemas.xmlsoap.org/wsdl/soap/" xmlns:soapenc=
"http://schemas.xmlsoap.org/soap/encoding/"
xmlns:s0=
"http://www.eraserver.net/webservices" targetNamespace=
"http://www.eraserver.net/webservices" xmlns=
"http://schemas.xmlsoap.org/wsdl/">
<types>
<s:schema attributeFormDefault="qualified" elementFormDefault="qualified">
<targetNamespace="http://www.eraserver.net/webservices">
<s:element name="DoMath">
<s:complexType>
<s:sequence>
<s:element name="Value1" type="s:double"/>
<s:element name="Value2" type="s:double"/>
</s:sequence>
</s:complexType>
</s:element>
<s:element name="DoMathResponse">
<s:complexType>
<s:sequence>
<s:element name="DoMathResult" type="s:double"/>

```

Listing 16.1 Continued

```
</s:sequence>
</s:complexType>
</s:element>
<s:element name="double" type="s:double"/>
</s:schema>
</types>
<message name="DoMathSoapIn">
  <part name="parameters" element="s0:DoMath"/>
</message>
<message name="DoMathSoapOut">
  <part name="parameters" element="s0:DoMathResponse"/>
</message>
```

Notice that, along with some general descriptive information at the top of Listing 16.1, there is a `<types>` section that describes the method available (`DoMath`) and it's input values (`Value1` and `Value2`) as well as the return information (`DoMathResponse`) and the associated output values (`double`). Finally, you see the call and response messages (`DoMathSoapIn` and `DoMathSoapOut`).

NOTE

WSDL messages include not only SOAP details, but also information on making HTTP calls to the component. You can learn more about the WSDL message by visiting www.W3C.org and searching for WSDL and SOAP.

There is a lot of information in Listing 16.1 and it might not all be familiar. Luckily, programmers using Microsoft's Web Services will not need to be too concerned with the details of WSDL messages. The only thing you need to understand is that, "under the covers," components you publish for use via Web Services will be able to automatically use WSDL in order to make sure they can properly communicate with all other SOAP-compliant servers.

The SOAP Request Message

Once the requesting machine has received a WSDL message, it has the information needed to compose and send a properly formatted SOAP request asking the target server to invoke a method and return the results. Using the WSDL shown in Listing 16.1, Server A could format a SOAP request that would look like the one in Listing 16.2.

Listing 16.2 SAMPLESOAP.REQUEST—A Sample SOAP Request to Invoke a Method and Return Results

```
<?xml version="1.0"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xs="http://www.w3.org/2000/10/XMLSchema" xmlns:soap=
  "http://schemas.xmlsoap.org/soap/envelope">
```

Listing 16.2 continued

```

<soap:Body>
  <DoMath xmlns="http://www.eraserver.net/webservices">
    <Value1>12</Value1>
    <Value2>12</Value2>
  </DoMath>
</soap:Body>
</soap:Envelope>

```

Listing 16.2 shows that Server A is sending a `<soap:Body>` that requests the `DoMath` method and passes two inputs (`Value1` and `Value2`) both set to “12.”

NOTE

It should also be noted that some additional header information (not shown here) is also sent with the request. This HTTP header data is important, but is not of special interest in this example.

The SOAP Response Message

Once the target server receives a properly formatted SOAP request, it can process the request and return the results of the method call. Listing 16.3 shows a sample SOAP response message that matches the SOAP request shown in Listing 16.2.

Listing 16.3 SAMPLESOAPRESPONSE—An Example SOAP Response Message to a SOAP Request

```

<?xml version="1.0"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" xmlns:soap=
  "http://schemas.xmlsoap.org/soap/envelope">
  <soap:Body>
    <DoMathResponse xmlns="http://www.eraserver.net/webservices">
      <DoMathResult>144</DoMathResult>
    </DoMathResponse>
  </soap:Body>
</soap:Envelope>

```

In Listing 16.3, you can see the `<DoMathResult>` element is set to “144,” the result of the math operation performed on the inputs passed in the SOAP request message shown in Listing 16.2.

NOTE

Again, some additional HTTP header information is sent with the response, but it is not shown in listing 16.3 since it is not of vital interest to the discussion.

Now that you understand how WSDL and SOAP messages are composed and passed between machines, it is time to discuss the details of publishing and consuming Microsoft Web Services.

Publishing and Consuming Web Services

There are a number of ways to publish and consume SOAP messages in order to support Microsoft's Web Services model. One way would be to use low-level TCP/IP sockets programming to format, send and receive, and parse XML messages of the type shown in Listings 16.1, 16.2, and 16.3 earlier in this chapter. This is a very effective way to take advantage of SOAP services, but is not very efficient. It requires programmers to know a great deal of information about low-level HTTP protocols as well as a deep understanding of the SOAP protocol and XML parsing.

In order to make it easier for Web developers to publish and consume components via SOAP Web Services, ASP.NET includes several class libraries that already understand the SOAP protocol. These classes simplify the process of creating and consuming Web Services on the ASP.NET platform.

In this section of the chapter, you'll learn the basics of publishing and consuming Web Services. In the two chapters that follow, you'll learn the details behind creating powerful Web Services components and Web Services clients.

Publishing Web Services

The process of creating a component and publishing it as a Web Service available via the Internet is very simple within ASP.NET. All you need to do is create a special document, an Active Server Method Extension file (ASMX), and place it in a publicly available Web on your server. Once you do this, any SOAP-compliant client can call into your server, receive the properly formatted WSDL and then make requests to your Web server and get back the requested results.

You do not need to actually compose any WSDL or create any SOAP messages yourself—ASP.NET will handle all this for you. In addition, the ASP.NET runtime will even automatically produce online documents that describe your Web Service and help anyone attempting to consume your service understand just what parameters they need to supply and what values they can expect in return.

As an example, consider that you want to publish a class called `sample` that contains a single method called `DoMath` that accepts two inputs of the type `double`, performs a multiplication operation on the inputs, and returns a single output of the type `double` that is the result of the math operation. Using ASP.NET, you could create this class in an ASMX file that would look like the one shown in Listing 16.4.

Listing 16.4 SAMPLE.ASMX—A Sample ASMX File to Publish a Web Service

```
<%@ webservice class="sample" language="vb" %>  
  
imports System
```

Listing 16.4 continued

```
imports System.Web.Services

public class sample : Inherits WebService

<WebMethod()> _
    public function DoMath(ByVal Value1 as double, ByVal Value2 as double)
        as double

    return(Value1*value2)

end function

end class
```

As you can see from Listing 16.4, the ASP.NET ASMX file format looks like a combination of an ASPX page and a simple VB Class component. The header at the top of the page indicates this will be a Web Service document. The document also shows an import of the `System.Web.Services` class library. This library contains the `WebServer` class that is used as the base class that the “sample” class inherits. It is this base class that understands how to communicate with remote callers using the SOAP protocol.

The rest of the page contains a typical class and method, coded the same way you would code any VB class. The only added code that you need to include is the `<WebMethod()>` attribute on the function you wish to publish. When adding this one attribute make sure that ASP.NET will allow remote callers to access this function.

By simply placing this ASMX file in your Web, you are now publishing a Web Service. You learn more about how this file behaves at runtime in Chapter 17 (“Publishing Web Services with ASP.NET”).

Consuming Web Services

Just as there are a number of ways to publish a SOAP-compliant Web Service, there are also several different ways to consume an existing Web Service. Again, you could use low-level programming to compose a SOAP request message and receive and parse the results for display. This would be very tedious, but would give you the highest degree of control over the entire process.

You can also use a special utility (WSDL.EXE) that ships with ASP.NET that would automatically create a special class to handle the SOAP conversation for you. All you would need to do is invoke the new class and program against the remote Web Service as you would any locally installed component. You learn how to use the WSDL.EXE in Chapter 18 (“Consuming Web Services with ASP.NET”).

You can also use simple HTML to access remote Web Services. This can be done with a standard HTML POST FORM or from a simple HREF in an anchor tag. For

example, you can call the DoMath method described in Listing 16.4 using the following single anchor tag on an HTML page:

```
<a href="sample.asmx/DoMath?value1=12&value2=12">DoMath</a>
```

You could also create a simple HTML POST FORM that allows users to enter their own data for Value1 and Value2 and then calls the Web Service when the submit button is pressed on the form. Listing 16.5 shows what this would look like on an HTML page.

Listing 16.5 HTMLCALLER.HTM—A Sample HTML POST FORM Calling a Web Service

```
<html>
<body>

<h2>HTML SOAP Caller Example</h2>
<hr />

<h3>HTML FORM</h3>
<form method="post" action="sample.asmx/DoMath">
    Value1: <input value=12 name="value1" size="10" /><br />
    Value2: <input value=12 name="value2" size="10" /><br />
    <input type="submit" value="DoMath" />
</form>

</body>
</html>
```

Pressing the anchor link mentioned previously or pressing the submit button in Listing 16.5 results in a call to the DoMath Web Service with ASP.NET automatically returning the XML message with the result (see Figure 16.2).

While very easy, using HTML pages to call Web Services always results in a return of plain XML data. In Chapter 18, you learn how to use the power of ASP.NET to create specialized server-side classes to accept the plain XML and return it to your ASP.NET applications as simple data that can be incorporated into standard ASP.NET documents and classes.

Other Important Issues

Now that you understand the basics of creating SOAP-based Web Services and how to publish and consume them, it is valuable to discuss several other important issues concerning public Web Services. The issues covered in this section of the chapter are

- Security
- State Management
- Transaction Management

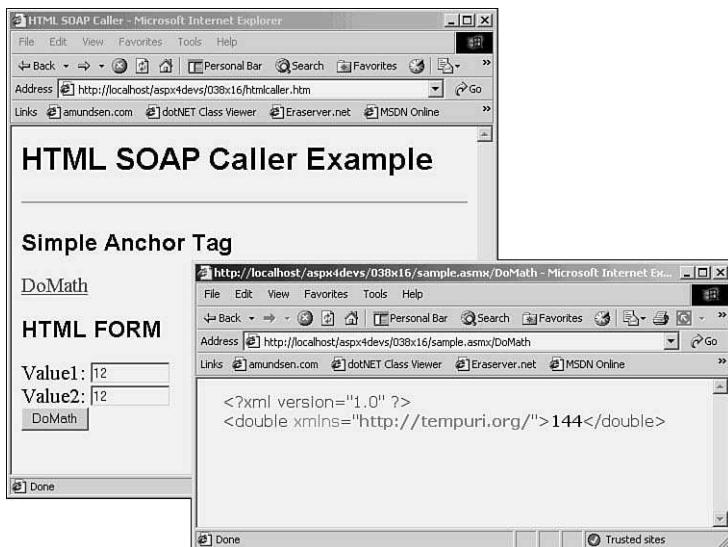


Figure 16.2

The XML message returned by an ASP.NET Web Service.

While all these issues are important, they do not necessarily apply to all Web Service scenarios. However, by addressing them here, you will have adequate information to consider them when you design and deploy your ASP.NET Web Services.

Security

The SOAP protocol has no built-in security mechanism or encryption scheme. These details are covered by existing standards including the use of standard security schemes, such as Basic and NTLM as well as secured sockets layer (SSL) for encryption.

For example, to create secured Web Services that require users to log in before they can access the service, all you need to do is place the ASMX page in a folder that has Internet Information Server security settings turned on. Then, when users attempt to access the Web Service method, they need to also pass username and password credentials.

For example, the code in Listing 16.6 shows how to set the username, password, and domain values before calling the Web Service method on a proxy class generated by the WSDL.EXE utility.

Listing 16.6 SECURESOAP.ASPX—Accessing a Secured SOAP Web Service

```
sub Page_Load(sender as object, args as EventArgs)
    dim oSample as Sample = new Sample()
```

Listing 16.6 continued

```
with oSample
    .username="aspx4devs"
    .password="smart"
    .domain="amundsen"
    results.Text = .DoMath(12,12)
end with

oSample = Nothing

end sub
```

State Management

By default, Web Services are stateless routines. All calls to a Web Service are single-state calls. However, you can optionally enable session state for Web Services using a special attribute setting on the `WebMethod`. This setting, called `EnableSession`, can be set to either TRUE or FALSE.

Listing 16.7 shows an example Web Service that allows users to read and write items into the remote server's session store.

Listing 16.7 SESSIONSERVICE.ASMX—An Example Web Service That Supports Session State

```
<%@ WebService class="SessionService" language="vb" %>

Option Strict Off

imports System
imports System.web.Services

public class SessionService : Inherits WebService

    <WebMethod(EnableSession:=true)> public function _
        WriteItem(byval Name as string, byval Value as string) as boolean
            try
                Session(Name) = Value
                return(true)
            catch
                return(false)
            end try
        end function

    <WebMethod(EnableSession:=true)> public function _
        ReadItem(byval Name as string) as string
```

Listing 16.2 continued

```

    return(Session(Name))

end function

end class

```

Transaction Management

You can also create Web Services that support Microsoft transaction Services. This allows you to build Web Services that can safely perform actions against multiple databases and still properly restore them if one or more of the databases reports an error. To support transaction services in your Web Service, you need to add the TransactionOption attribute to the WebMethod.

Listing 16.8 shows an example of a Web Service method that attempts to update two different databases and provides transaction protection.

Listing 16.8 TRANSACTIONSERVICE.ASMX—An Example of Supporting Transactions in a Web Service

```

<%@ WebService class="TransactionService" language="vb" %>

imports System
imports System.web.Services
imports System.EnterpriseServices

public class TransactionService : Inherits WebService

    <WebMethod( TransactionOption:=TransactionOption.Required)> _
    public function updateDatabases(byval updatedSet as DataSet) as boolean

        try
            updatePrimaryDb(updatedSet)
            updateBackupDb(updatedSet)
            return(true)
        catch
            return(false)
        end try

    end function

end class

```

Summary

In this chapter, you learned what Web Services are and how ASP.NET implements the SOAP standard. You also learned the history of SOAP Web Services and the current status of the SOAP standard at the W3C organization.

In addition, you learned the basics of how to publish and consume Web Services. You also learned how to deal with other important issues including security, state management, and transactions.

In the following two chapters, you learn more details on publishing and consuming Web Services using ASP.NET.

CHAPTER 17

Publishing Web Services with ASP.NET

Now that you have learned the general concept of ASP.NET Web Services from Chapter 16 (“Understanding the Web Service Model”), you’re ready to delve into the details of creating Web Services with ASP.NET. This includes a review of a typical ASP.NET Web Services code template, the process of defining and coding web methods, exploring the details of the default web page presented by ASP.NET when users call your Web Service document, and how to modify the contents of the WSDL message using Web Service and web method attributes.

After covering the details of a typical Web Service that returns simple values, you learn how to use Web Services to return more complex values such as array lists, custom classes, and hierarchical DataSets.

Once you’ve completed this chapter, you’ll know how to create Web Services that return a wide range of simple and complex data types.

The Basics of Creating Web Services with ASP.NET

As you saw in the preceding chapter, creating cross-platform Web Services with ASP.NET is relatively easy. All you need to do is add at least one ASMX file to a public web on your server and code one or more methods in the class and mark them with the `WebMethod` attribute to publish them on the public server.

In this part of the chapter you learn the basics of creating Web Services including the contents of a typical ASMX template, how to add public web methods to the template, how to

understand the contents of the default help web page generated by ASP.NET, and how to modify the output of the WSDL file using web method and Web Service attributes.

A Typical ASMX Template

Every Web Service you host in ASP.NET starts with an ASMX template file. This file, much like its HTML cousin the ASPX file, contains at least one directive followed by familiar code that defines a class that will contain the public web methods.

Following is a typical `webService` directive that must appear at the top of the document.

```
<%@ webService  
    class="wsBasic"  
    language="vb"  
    description="a basic web service"  
%>
```

The `class` attribute tells ASP.NET which class in this document contains the web methods. This must be identical to one of the public classes on the page. If it's not, you'll receive a compile time error. You also need to include the `Language` attribute to indicate which compiler ASP.NET should use on the page. The `Description` attribute is optional, but it is a good idea to include it in all your ASMX files.

The remainder of the file looks like any ASP.NET compiled component. You need to import two class libraries for every ASMX file: the `System` class library and the `System.Web.Services` class library. The first is needed to support general methods and properties. The second one is needed to support SOAP-based Web Services in your class.

NOTE

Technically, you do not need to import any class libraries as long as you use complete names when referring to classes and types. However, your code will be much easier to read and maintain if you use import statements.

Finally, you need to declare at least one class in your file that has the same name as the `CLASS` attribute in your page directive. It also needs to inherit from the `WebService` base class that is part of the `System.Web.Services` class library. The following snippet shows a more complete ASMX template that meets all the requirements mentioned so far.

```
<%@ webService  
    class="wsBasic"  
    language="vb"  
    description="a basic web service"  
%>  
  
imports System
```

```
imports System.Web.Services

public class wsBasic : Inherits WebService

end class
```

At this point the template shown in the first snippet will successfully compile, but does not yet publish any method for remote execution via Web Services. If you plan to create a number of Web Services, you might want to save a file similar to the one shown in the first snippet as your starter template to save you time and reduce the chance of errors.

Building Public WebMethods

Now that you have a basic ASMX template to start with, you're ready to add one or more methods to the class that will be published to the Internet. All you need to do is create a Visual Basic function that returns some type of value and mark it as a public web method. The following snippet, WSBASIC.ASMX, shows a very simple web method that returns a string to the caller.

```
<WebMethod()> public function SayHello() as string

    return("This is a simple web method")

end function
```



NOTE

It is actually possible to create a WebMethod using a SUB instead of a FUNCTION. However, since SUBS do not return any values, they are not very valuable when creating Web Services.

You'll notice that the only thing unusual about the Visual Basic function shown in the preceding snippet is that it is marked with a `WebMethod` attribute. This attribute is required if you wish to allow others to call your method using SOAP services.

You can create functions that take any number and type of input arguments and return almost any type of value. There is also virtually no limit to the type of code you can execute within the web method. The following snippet shows a slightly more interesting web method that takes two arguments and returns a simple numeric value.

```
<WebMethod()> public function DoMath(ByVal Value1 as double, _
    ByVal Value2 as double) as double

    return(Value1*value2)

end function
```

It is important to point out that you can have any number of methods you wish in your Web Service class. You can also have any number of methods in the class that are not published as web methods. They can be private, shared, or any other type of methods. In addition, you can have other non-Web Service classes in the same file as the Web Service class. Of course you can also import any other precompiled custom class libraries you need, too.

Exploring the Default SDLHelpGenerator Pages

Once you have a fully functional ASMX file and you've placed it in a public web, users can access this page and execute your web methods. However, unless they know the names of the methods and their parameters exactly, calling your Web Services can be a frustrating hit-and-miss experience.

To make all this easier, ASP.NET can automatically generate a help page for users who want to learn more about your Web Services. Users simply need to make a call directly to the ASMX file in your web. For example, Figure 17.1 shows the results of requesting the WSBASIC.ASMX file shown earlier in this chapter.

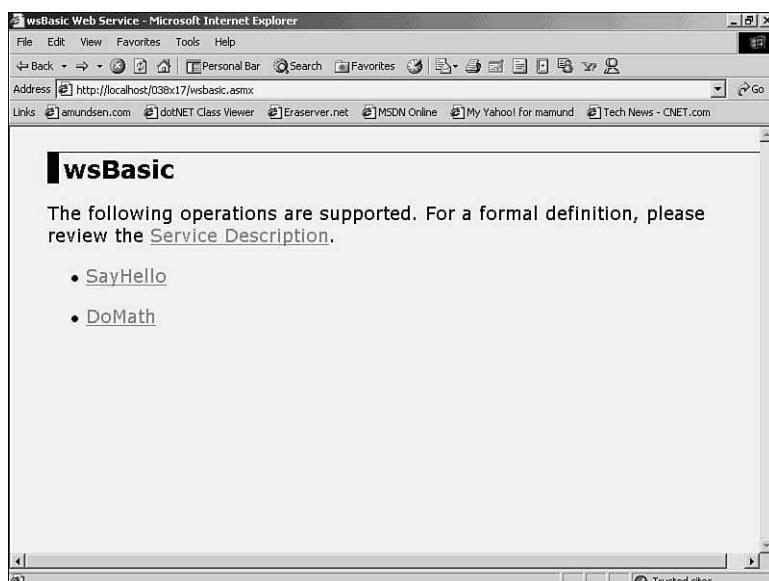
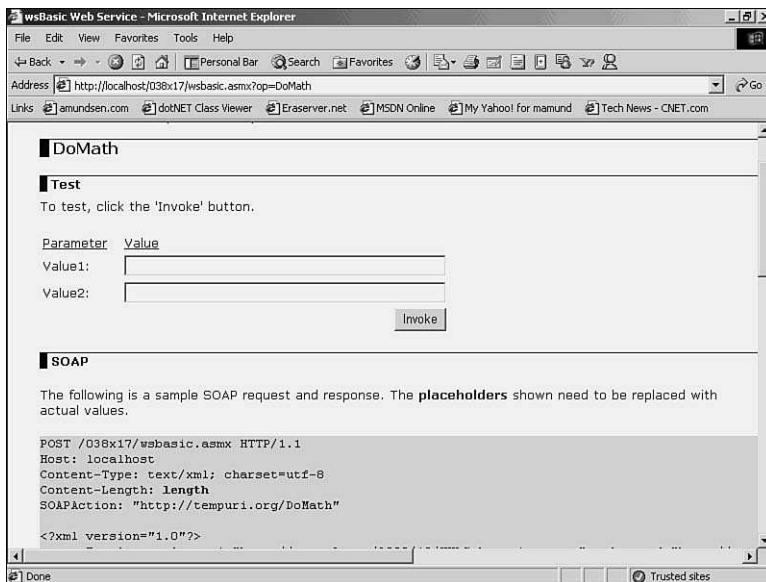


Figure 17.1

The results of requesting the WSBASIC.ASMX document from a browser.

You can get additional information on each method by clicking on the name in the help page. Figure 17.2 shows the added help for the “DoMath” method of the WSBASIC.ASMX file. You can see that along with detailed documentation on the web method, ASP.NET even provides a sample input form so that users can test the web method.

**Figure 17.2**

A detailed help and input form generated by ASP.NET.

NOTE

The document generated by ASP.NET can be replaced by your own customized help document. All you need to do is build one based on the DEFAULTWSDLHELPGENERATOR.ASPX file installed on your Web server and then register the new page in using the <wsdlHelpGenerator> element in your WEB.CONFIG file.

Adding WebMethod and WebService Attributes

You can control the output of the WSDL file and the help file as well as affect the run-time performance of your Web Services by inserting additional attributes into the ASMX document. There are two different attribute sets you can use in ASMX files: the WebService attribute and the WebMethod attribute. In this section, you'll learn how to use these two attributes in your Web Service documents.

The WebService Attribute

The WebService attribute can be applied to the class definition in order to modify the WSDL message output by ASP.NET when callers request information about your Web Service. The WebService attribute allows you to set or get three values from the WSDL for your Web Service:

- Description: This is a text description that will appear in the WSDL document sent to the caller.

- Name: This is the SOAP name that will be published for your method. Use this if you want your published name to be different than the class name.
- Namespace: This is the SOAP namespace associated with your Web Service. By default, ASP.NET always uses the `http://tempuri.org` namespace unless you set this value.

The following code shows how you can apply these values to the `WebService` attribute in your ASMX document.

```
<WebService( _  
    Name:="wsAttrib", _  
    Namespace:="http://www.eraserver.net/webservices", _  
    Description:="Accepts two doubles, multiplies them, returns a double" _  
)> public class _  
wsAttrib : Inherits WebService
```

The WebMethod Attribute

You already know that the `WebMethod` attribute is used to mark one or more methods in the web class as a public method. However, you can also insert additional values in the `WebMethod` attribute to control both the runtime behavior and the help document associated with the web methods.

There are six different `WebMethod` values you can work with:

- `BufferResponse`: Set this value to false if you want to turn off buffering.
- `CacheDuration`: By default, return values are not cached in memory. If the return values do not change often, you can increase performance of your web method by caching the output.
- `Description`: This description will appear on the help page.
- `EnableSession`: Set this value to true to enable session state for your web method.
- `MessageName`: Set this value to change the SOAP message to something other than the method name.
- `TransactionOption`: Set this value to allow your method to support distributed transaction services.

The following snippet shows WSATTRIB.ASMX, an example of how you can apply these values to a web method in your ASMX file.

```
<WebMethod( _  
    CacheDuration:=0, _  
    EnableSession:=true, _  
    BufferResponse:=true, _  
    MessageName:="DoMath", _  
    Description:="Returns the results of multiplication" _  
)>public function _  
DoMath(ByVal Value1 as double, ByVal Value2 as double) as double
```

Creating Other Web Service Examples

While it is common to return simple values from a Web Service (a single string or number), many of the Web Services you will want to build will require the return of more complex values such as arrays, custom class objects or complex datasets. In this section, you'll learn how to design and implement Web Services that return complex values.

Returning ArrayLists

In some cases, you'll want to create Web Services that can return a simple array of values. This is actually very easy to do with ASP.NET Web Services. All you need to do is create an array in the Web Service file and configure it at the return value of the web method.

Listing 17.1 shows an ASMX file that returns an ArrayList.

Listing 17.1 WSARRAYLIST.ASMX—A Web Service That Returns an Array of Values

```
<%@ WebService language="vb" class="wsArrayList" %>

imports System
imports System.Web.Services
imports System.Collections

public class wsArrayList : Inherits WebService

    <WebMethod()> public function getTownList() as ArrayList

        dim aryTowns as ArrayList = new ArrayList()

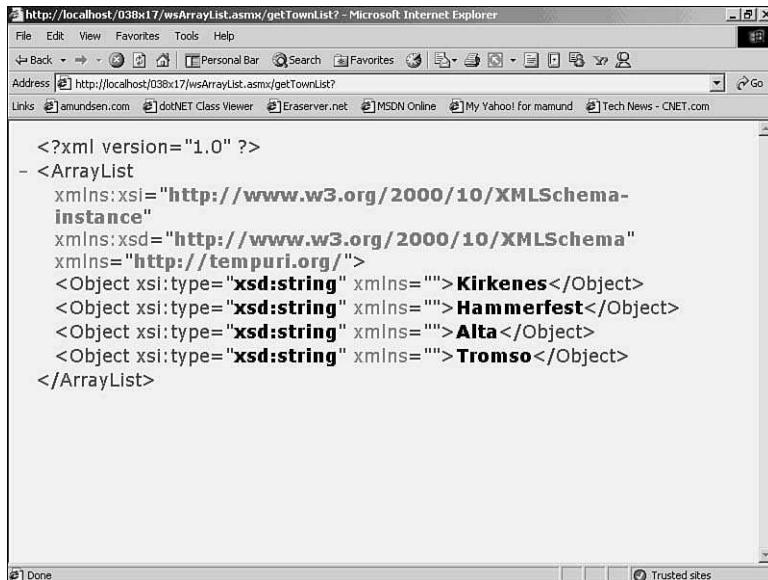
        with aryTowns
            .Add("Kirkenes")
            .Add("Hammerfest")
            .Add("Alta")
            .Add("Tromso")
        end with

        return(aryTowns)

    end function

end class
```

You'll notice that there are no special settings you need to use to return the ArrayList object. Figure 17.3 shows the results of calling the getTownList web method.

**Figure 17.3**

Results of calling the `getTownList` web method.

As you can see from Figure 17.3, ASP.NET will automatically convert the `ArrayList` object to an XML list. When you send this list to another ASP.NET page, that page will be able to convert the XML list back into an `ArrayList` object and display it on a page.

Returning Custom Classes

There are also times when you want to return a custom object to the caller. For example, you might have an object that represents customers in your business. This object might contain a number of properties and methods associated with your customers. You can use ASP.NET web methods to pass this customer object from one machine to another.

Listing 17.2 shows how this can be done with ASP.NET Web Services.

Listing 17.2 WSCUSTOMCLASS.ASMX—Returning a Custom Class with ASP.NET Web Services

```

<%@ WebService language="vb" class="wsCustomClass" %>

imports System
imports System.Web.Services

```

Listing 17.2 continued

```
public class wsCustomClass : Inherits WebService

<WebMethod()> public function getCustomer() as CustomerClass

    dim objCustomer as CustomerClass = new CustomerClass()

    with objCustomer
        .ID=13
        .FirstName = "Jorn"
        .LastName = "Wibe"
    end with

    return(objCustomer)

end function

end class

public class CustomerClass

    private _ID as integer
    private _FirstName as string
    private _LastName as string
    public property FirstName as string
        get
            FirstName = _FirstName
        end get
        set
            _FirstName = value
        end set
    end property

    public property LastName as string
        get
            LastName = _LastName
        end get
        set
            _LastName = value
        end set
    end property

    public property ID as integer
        get
            ID = _ID
        end get
    end property

```

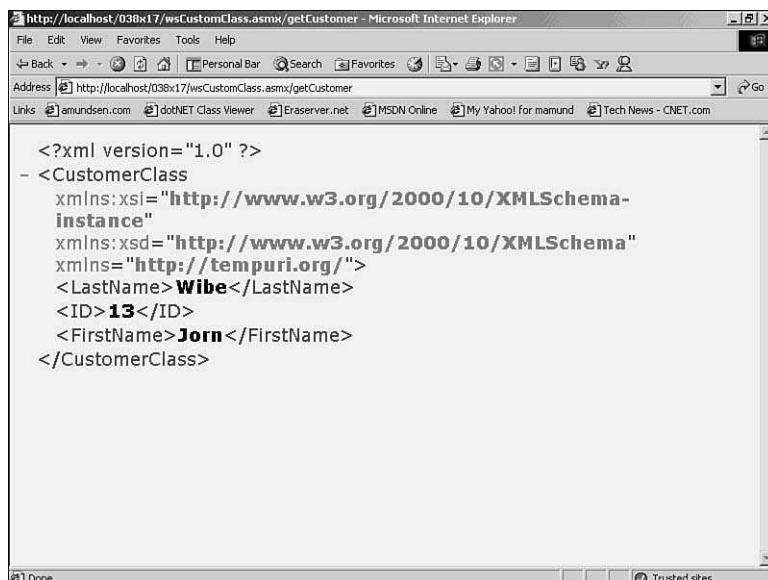
Listing 17.2 continued

```
end get
set
    _ID = value
end set
end property

public function FullName() as string
    return(_FirstName & " " & LastName)
end function

end class
```

Although this is a long example, it illustrates several important points. First, you'll notice that this ASMX document contains more than one class. Also, if you compare the list of *CustomerClass* members in Listing 17.2 to the data returned in Figure 17.4, you'll notice that the *FullName* function in Listing 17.2 did not appear in the data returned by the web method. This is an important point. When working with custom classes, web methods will only return property values, not the results of functions.

**Figure 17.4**

Results of requesting a custom class. Notice that only properties are returned.

Returning Complex DataSets

The last example in this chapter shows you how you can return hierarchical DataSets from your web methods. Once again, ASP.NET can automatically marshal complex ADO.NET DataSet objects as XML for a SOAP web method. The calling machine will then be able to consume the XML and pour it back into a DataSet object for use in the local application.

Listing 17.3 shows how to create a Web Service ASMX file that returns a hierarchical DataSet and Figure 17.5 shows the XML results that are returned.

Listing 17.3 WSDATASET.ASMX—An ASMX File That Returns a Complex ADO.NET DataSet

```
<%@ WebService language="vb" class="wsDataSet" %>

imports System
imports System.Web.Services
imports System.Data
imports System.Data.SqlClient

public class wsDataSet

<WebMethod()> public function getPubsData() as DataSet

    dim sConn as string = "server=localhost;database=pubs;uid=sa;pwd=;"
    dim sPubs as string = "SELECT Pub_ID, Pub_Name FROM Publishers"
    dim sTitles as string = "SELECT Title_ID, Title, Pub_ID FROM Titles"

    dim ds as DataSet = new DataSet()

    dim cn as SqlConnection = new SqlConnection(sConn)
    dim daPubs as SqlDataAdapter = new SqlDataAdapter(sPubs,cn)
    dim daTitles as SqlDataAdapter = new SqlDataAdapter(sTitles,cn)

    daPubs.Fill(ds,"Pubs")
    daTitles.Fill(ds,"Titles")

    ds.Relations.Add("PubsTitles", _
        ds.Tables("Pubs").Columns("Pub_ID"), _
        ds.Tables("Titles").Columns("Pub_ID"))

    return(ds)

end function

end class

<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://tempuri.org/">
```

Listing 17.3 continued

```
- <xsd:schema id="NewDataSet" targetNamespace="" xmlns="" xmlns:xsd=
➥ "http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com
➥ :xml-msdata">
- <xsd:element name="NewDataSet" msdata:IsDataSet="true">
- <xsd:complexType>
- <xsd:choice maxOccurs="unbounded">
- <xsd:element name="Pubs">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element name="Pub_ID" type="xsd:string" minOccurs="0" />
<xsd:element name="Pub_Name" type="xsd:string" minOccurs="0" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
- <xsd:element name="Titles">
- <xsd:complexType>
- <xsd:sequence>
<xsd:element name="Title_ID" type="xsd:string" minOccurs="0" />
<xsd:element name="Title" type="xsd:string" minOccurs="0" />
<xsd:element name="Pub_ID" type="xsd:string" minOccurs="0" />
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
- <xsd:unique name="Constraint1">
<xsd:selector xpath=".//Pubs" />
<xsd:field xpath="Pub_ID" />
</xsd:unique>
- <xsd:keyref name="PubsTitles" refer="Constraint1">
<xsd:selector xpath=".//Titles" />
<xsd:field xpath="Pub_ID" />
</xsd:keyref>
</xsd:element>
</xsd:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"
➥ xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <NewDataSet xmlns="">
- <Pubs diffgr:id="Pubs1" msdata:rowOrder="0">
<Pub_ID>0736</Pub_ID>
<Pub_Name>New Moon Books</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs2" msdata:rowOrder="1">
<Pub_ID>0877</Pub_ID>
<Pub_Name>Binnet & Hardley</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs3" msdata:rowOrder="2">
```

Listing 17.3 continued

```
<Pub_ID>1389</Pub_ID>
<Pub_Name>Algodata Infosystems</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs4" msdata:rowOrder="3">
<Pub_ID>1622</Pub_ID>
<Pub_Name>Five Lakes Publishing</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs5" msdata:rowOrder="4">
<Pub_ID>1756</Pub_ID>
<Pub_Name>Ramona Publishers</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs6" msdata:rowOrder="5">
<Pub_ID>9901</Pub_ID>
<Pub_Name>GGG&G</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs7" msdata:rowOrder="6">
<Pub_ID>9952</Pub_ID>
<Pub_Name>Scootney Books</Pub_Name>
</Pubs>
- <Pubs diffgr:id="Pubs8" msdata:rowOrder="7">
<Pub_ID>9999</Pub_ID>
<Pub_Name>Lucerne Publishing</Pub_Name>
</Pubs>
- <Titles diffgr:id="Titles1" msdata:rowOrder="0">
<Title_ID>BU1032</Title_ID>
<Title>The Busy Executive's Database Guide</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles2" msdata:rowOrder="1">
<Title_ID>BU1111</Title_ID>
<Title>Cooking with Computers: Surreptitious Balance Sheets</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles3" msdata:rowOrder="2">
<Title_ID>BU2075</Title_ID>
<Title>You Can Combat Computer Stress!</Title>
<Pub_ID>0736</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles4" msdata:rowOrder="3">
<Title_ID>BU7832</Title_ID>
<Title>Straight Talk About Computers</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles5" msdata:rowOrder="4">
<Title_ID>MC2222</Title_ID>
<Title>Silicon Valley Gastronomic Treats</Title>
<Pub_ID>0877</Pub_ID>
```

Listing 17.3 continued

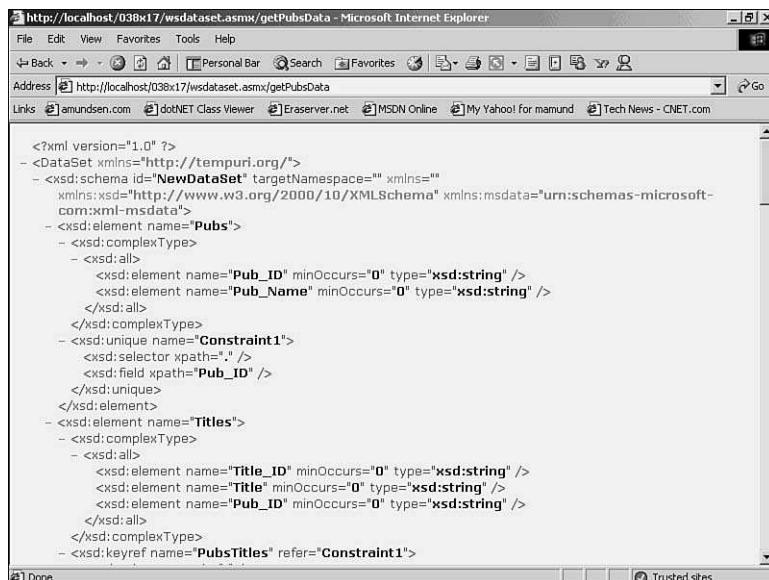
```
</Titles>
- <Titles diffgr:id="Titles6" msdata:rowOrder="5">
<Title_ID>MC3021</Title_ID>
<Title>The Gourmet Microwave</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles7" msdata:rowOrder="6">
<Title_ID>MC3026</Title_ID>
<Title>The Psychology of Computer Cooking</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles8" msdata:rowOrder="7">
<Title_ID>PC1035</Title_ID>
<Title>But Is It User Friendly?</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles9" msdata:rowOrder="8">
<Title_ID>PC8888</Title_ID>
<Title>Secrets of Silicon Valley</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles10" msdata:rowOrder="9">
<Title_ID>PC9999</Title_ID>
<Title>Net Etiquette</Title>
<Pub_ID>1389</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles11" msdata:rowOrder="10">
<Title_ID>PS1372</Title_ID>
<Title>Computer Phobic AND Non-Phobic Individuals: Behavior
➥Variations</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles12" msdata:rowOrder="11">
<Title_ID>PS2091</Title_ID>
<Title>Is Anger the Enemy?</Title>
<Pub_ID>0736</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles13" msdata:rowOrder="12">
<Title_ID>PS2106</Title_ID>
<Title>Life Without Fear</Title>
<Pub_ID>0736</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles14" msdata:rowOrder="13">
<Title_ID>PS3333</Title_ID>
<Title>Prolonged Data Deprivation: Four Case Studies</Title>
<Pub_ID>0736</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles15" msdata:rowOrder="14">
```

Listing 17.3 continued

```

<Title_ID>PS7777</Title_ID>
<Title>Emotional Security: A New Algorithm</Title>
<Pub_ID>0736</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles16" msdata:rowOrder="15">
<Title_ID>TC3218</Title_ID>
<Title>Onions, Leeks, and Garlic: Cooking Secrets of the
➥Mediterranean</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles17" msdata:rowOrder="16">
<Title_ID>TC4203</Title_ID>
<Title>Fifty Years in Buckingham Palace Kitchens</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
- <Titles diffgr:id="Titles18" msdata:rowOrder="17">
<Title_ID>TC7777</Title_ID>
<Title>Sushi, Anyone?</Title>
<Pub_ID>0877</Pub_ID>
</Titles>
</NewDataSet>
</diffgr:diffgram>
</DataSet>

```

**Figure 17.5**

The results of requesting a complex DataSet from a web method.

Summary

In this chapter you learned the details of creating typical ASMX templates to publish Web Services. You can modify the WSDL output of your Web Service by using the `WebService` and `WebMethod` Attributes within your ASMX file. Arrays, custom classes, and hierarchical ADO.NET DataSet objects are among the more complex data your Web Services can return.

In the next chapter, you will learn how to use ASP.NET and SOAP to create powerful client applications that can consume these and other Web Services.

CHAPTER 18

Consuming Web Services with ASP.NET

In the preceding chapter, you learned how to create ASMX files that publish ASP.NET Web Services. In this chapter, you learn how to use ASP.NET to create applications that can communicate with the Web Service files published in ASP.NET solutions.

First, you go through a quick review of the Web Service Description Language (WSDL). You learn some of the details of the HTTP and SOAP protocols that you can use to send successful Web Service requests to remote machines.

Next, you learn how to build simple HTTP clients to communicate with Web Services. You also learn how to create SOAP proxy classes in order to allow your ASP.NET client to communicate directly to remote Web Services.

Review of WSDL Contracts

Before jumping into the details of building the various clients that can communicate with Web Services on remote machines, it is valuable to spend some time reviewing the Web Service Description Language (WSDL). WSDL is the XML-based document that is issued by Web Services hosts. Studying its contents will help you gain a greater understanding of how Web Services work and how you can build effective and efficient Web Service clients.

The WSDL document is an XML-formatted file that contains all the details about the remote component and its methods and arguments. The document also contains information about the machine hosting the Web Services as well as descriptive information about the methods published by the Web Service.

One of the more powerful features of ASP.NET is that programmers do not need to have a deep understanding of the WSDL used to communicate between machines. In fact, if you use Microsoft's Visual Studio.NET, you will be able to build both Web Service publishing and consuming solutions without ever even looking at WSDL contract files or any of the XML messages that are sent between the two machines. However, knowing the file's contents and its format will help you more easily solve problems and also give you the chance to modify the file's contents to improve the quality of your SOAP messages between machines.

The current format of the WSDL document is divided into five main information types:

- Services: Describes the actual “entry points” for this Web Service for each supported protocol.
- Bindings: Describes the port types used for each protocol that this Web Service supports.
- Port Types: Describes the collection of related input and output messages for each method call.
- Messages: Describes the input and output messages this Web Service supports.
- Schema: Describes the low-level methods, arguments, and return values.

The following sections describe the major portions of the WSDL document in greater detail. All the descriptions here are based on a simple Web Service that has a single method that accepts a lowercase string and returns an uppercase version of the same string. Listing 18.1 shows the complete ASMX file for this Web Service.

Listing 18.1 WSSTRINGS.ASMX—A Sample ASP.NET Web Service

```
<%@ webservice
  class="wsStrings"
  language="vb"
  description="string handling web service"
%>

imports System
imports System.Web.Services

public class wsStrings : Inherits WebService

  <WebMethod()> public function MakeUpper(ByVal Data as string) as string
    return(Data.ToUpper())

  end function

end class
```

Services and Bindings

The top level information in the WSDL document describes the “entry points” for the Web Service. In ASP.NET, the entry point is usually the name of the ASMX file.

However, you can override this by setting the Name value of the WebService attribute (see Chapter 16, “Understanding the Web Service Model,” for details on the WebService attribute). By default, all ASP.NET Web Services support three entry points: SOAP, HTTP-GET, and HTTP-POST. Listing 18.2 shows a sample <service> section in the WSDL document for the ASMX file shown in Listing 18.1.

Listing 18.2 WSSTRINGS.WSDL—Service Section from a WSDL Document

```
<service name="wsStrings">
  <port name="wsStringsSoap" binding="s0:wsStringsSoap">
    <soap:address location="http://localhost/aspx4devs/038x18/wsStrings.asmx" />
  </port>
  <port name="wsStringsHttpGet" binding="s0:wsStringsHttpGet">
    <http:address location="http://localhost/aspx4devs/038x18/wsStrings.asmx" />
  </port>
  <port name="wsStringsHttpPost" binding="s0:wsStringsHttpPost">
    <http:address location="http://localhost/aspx4devs/038x18/wsStrings.asmx" />
  </port>
</service>
```

Listing 18.2 shows that each of the three port elements has a “binding” attribute. This points to another entry in the WSDL document. The binding defines the details of how each protocol handles requests and sends responses.

For example, Listing 18.3 shows the three bindings for SOAP, HTTP-GET, and HTTP-POST. The wsStringsHttpGet binding shows that the input operation is a simple URL and the output is an HTTP body. This corresponds to the HTML page response that you get when you click on the following URL tag in a Web page:

```
<a href="wsStrings.asmx/MakeUpper?data=hammerfest>MakeUpper</a>
```

Listing 18.3 WSSTRINGS.WSDL—The Bindings Entries from a WSDL Document

```
<binding name="wsStringsSoap" type="s0:wsStringsSoap">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <operation name="MakeUpper">
    <soap:operation soapAction="http://tempuri.org/MakeUpper"
      style="document" />
    <input>
      <soap:body use="literal" />
    </input>
    <output>
      <soap:body use="literal" />
    </output>
  </operation>
</binding>
```

Listing 18.3 continued

```
<binding name="wsStringsHttpGet" type="s0:wsStringsHttpGet">
  <http:binding verb="GET"/>
  <operation name="MakeUpper">
    <http:operation location="/MakeUpper" />
    <input>
      <http:urlEncoded/>
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
</binding>
<binding name="wsStringsHttpPost" type="s0:wsStringsHttpPost">
  <http:binding verb="POST"/>
  <operation name="MakeUpper">
    <http:operation location="/MakeUpper" />
    <input>
      <mime:content type="application/x-www-form-urlencoded" />
    </input>
    <output>
      <mime:mimeXml part="Body" />
    </output>
  </operation>
</binding>
```

Notice that each binding element has a “type” attribute. This points to a PortType section that defines the set of messages and data types used for each entry point. These three features are covered in the next section of this chapter.

PortTypes, Messages, and Schema

The PortTypes sections define the set of input and output messages sent by the Web Service in order to support each entry point defined in the `<service>` section. There is one `<PortType>` element for each `<Binding>` element in the WSDL document.

Listing 18.4 shows the `<PortType>` sections for the WSDL document that match the Web Service file in Listing 18.1.

Listing 18.4 WSSTRINGS.WSDL—Sample PortType Sections from a WSDL Document

```
<portType name="wsStringsSoap">
  <operation name="MakeUpper">
    <input message="s0:MakeUpperSoapIn" />
    <output message="s0:MakeUpperSoapOut" />
  </operation>
</portType>
<portType name="wsStringsHttpGet">
  <operation name="MakeUpper">
```

Listing 18.4 continued

```

<input message="s0:MakeUpperHttpGetIn"/>
<output message="s0:MakeUpperHttpGetOut"/>
</operation>
</portType>
<portType name="wsStringsHttpPost">
  <operation name="MakeUpper">
    <input message="s0:MakeUpperHttpPostIn"/>
    <output message="s0:MakeUpperHttpPostOut"/>
  </operation>
</portType>

```

The `<PortType>` element describes each method available along with the input and output messages used for that method. The input and output message elements for each protocol type correspond to message entries in the document.

The `<message>` elements detail the number and type of the input arguments and output responses. As in the previous sections, there is an input and output message for each protocol that is supported by the Web Service. Listing 18.5 shows the message elements from the WSDL that match the code in Listing 18.1.

Listing 18.5 WSSTRINGS.WSDL—The Message Elements of a WSDL Document

```

<message name="MakeUpperSoapIn">
  <part name="parameters" element="s0:MakeUpper"/>
</message>
<message name="MakeUpperSoapOut">
  <part name="parameters" element="s0:MakeUpperResponse"/>
</message>
<message name="MakeUpperHttpGetIn">
  <part name="Data" type="s:string"/>
</message>
<message name="MakeUpperHttpGetOut">
  <part name="Body" element="s0:string"/>
</message>
<message name="MakeUpperHttpPostIn">
  <part name="Data" type="s:string"/>
</message>
<message name="MakeUpperHttpPostOut">
  <part name="Body" element="s0:string"/>
</message>

```

Each HTTP input message element contains the argument name and data type. The HTTP output message elements contain the same data, but notice that the output type is really a BODY for an HTTP message. This equates to an HTML document as the return value.

Both the input and output SOAP messages point to a parameter list and an entry in the schema section that defines the input and output parameters. Listing 18.6 shows the <schema> element of the WSDL document. This shows the details of the data types that SOAP uses to handle the input and output values for each method in the Web Service.

Listing 18.6 WSSTRINGS.WSDL—The Schema Elements of a WSDL Document

```
<types>
  <s:schema attributeFormDefault="qualified"
    elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
    <s:element name="MakeUpper">
      <s:complexType>
        <s:sequence>
          <s:element name="Data" nullable="true" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="MakeUpperResponse">
      <s:complexType>
        <s:sequence>
          <s:element name="MakeUpperResult" nullable="true" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <s:element name="string" nullable="true" type="s:string"/>
  </s:schema>
</types>
```

In Listing 18.6, you can see both the `MakeUpper` and `MakeUpperResponse` elements and the associated data names and types. The schema section is the last major section of the WSDL document.

Now that you understand the major parts of the WSDL document that are used to manage the Web Service communication between two machines, you’re ready to explore the various ways you can create client applications that call into these Web Services.

Creating HTTP Web Service Clients

As mentioned at the start of this chapter, there are a number of ways to create client applications that call Web Services. One of the simplest ways to access Web Services is to use basic HTTP-GET and HTTP-POST messages. HTTP calls always return an XML document body. Not all browsers can accept XML documents as return values, however Microsoft Internet Explorer is one of the browsers that is capable of handling XML documents.

NOTE

Although direct HTTP calling has limited uses in live applications, it is very valuable as a testing tool and in cases where you are using low-level programming at the sockets level to send and receive HTTP messages.

Using HTTP-GET to Retrieve Web Service Data

It is very easy to make a call directly to a Web Service using HTTP-GET. For example, a simple HTML ANCHOR tag (<a>) is a version of an HTTP-GET message. The following code snippet shows an example of using HTTP-GET to make a Web Service call.

```
<a href="wsStrings.asmx/MakeUpper?Data=hammerfest>MakeUpper</a>
```

Clicking on this link in a browser will make a call to the Web Service. Once the call is received, the hosting machine uses the HTTP-GET service definition in the WSDL document, deciphers the inputs, calls the requested method, formats the return value as an XML message, and sends it back to the browser. The following code snippet shows what the returned XML message looks like.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">HAMMERFEST</string>
```

Using HTTP-POST to Retrieve Web Service Data

The HTTP-GET method of testing a Web Service is easy, but sometimes you need to create a more sophisticated way to test the Web Service. A good way to create a test bed that allows you to easily enter different values to test the service is to create a familiar HTTP-POST FORM with a “submit” button.

To do this, all you need to do is create a standard HTML page that contains one or more inputs in an HTML FORM along with a SUBMIT button. Listing 18.7 shows an example HTML FORM that tests the Web Service from Listing 18.1.

Listing 18.7 WSSTRINGSFORM.HTM—An HTML FORM That Calls a Web Service

```
<html>
<body>

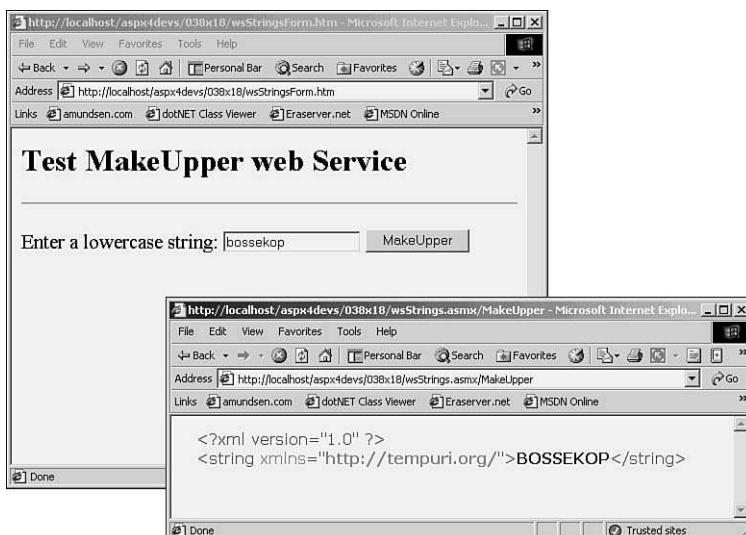
<h2>Test MakeUpper web Service</h2>
<hr />

<form action="wsStrings.asmx/MakeUpper" method="post">
    Enter a lowercase string:
    <input type="text" name="Data" />
    <input type="submit" value="MakeUpper" />
</form>
```

Listing 18.7 continued

```
</body>  
</html>
```

Listing 18.7 shows that the ACTION attribute of the HTTP-POST form must contain both the document URL and the method name you wish to execute. Also, the HTML FORM must contain INPUT elements that have a NAME attribute that matches the input names in the WSDL contract. Figure 18.1 shows what the HTML FORM looks like in the Microsoft Internet Explorer browser.

**Figure 18.1**

Testing the HTML FORM to call a Web Service.

When the document is loaded in a browser, all you need to do is enter some data in the input control and press the SUBMIT button. This will send the HTTP-POST message to the Web Service and the Web Service will use information in the WSDL document to parse the data, execute the method, and return an XML message to the caller. The return string will be identical to the return string sent using the HTTP-GET method.

Now that you know how to build simple test clients that call Web Services, you're ready to build client interface that you can use in production applications.

Creating SOAP Web Service Clients

ASP.NET makes it easy to build Web forms and components that can successfully call Web Services using the SOAP protocol, properly parse the SOAP return message, and present the resulting value for you to use in your component or Web form just as you would the return value from any standard component.

There are two basic steps to building a SOAP-aware client application. First, you need to create what is known as a SOAP Proxy class. This is a specially compiled component that knows how to format your method call into a valid SOAP message and how to parse an incoming SOAP return message.

Once you have created the special SOAP proxy class, you need to code a standard ASP.NET Web form or compiled component that can use the proxy class to send the calls to the target Web Service.

Generating a SOAP Proxy with WSDL.EXE

Step one in creating a SOAP-enabled ASP.NET Web form or component is creating a SOAP proxy class. Listing 18.8 shows what a typical SOAP proxy class looks like.

Listing 18.8 WSSTRINGS PROXY.VB—A Typical SOAP Proxy Class for a Web Service

```
Imports System
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization

Namespace aspx4devs

    Public Class <System.Web.Services.WebServiceBindingAttribute(
        Name:="wsStringsSoap", [Namespace]:="http://tempuri.org/")> WsStrings
        Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

        Public Sub New()
            MyBase.New
            Me.Url = "http://localhost/aspx4devs/038x18/wsStrings.asmx"
        End Sub

        Public Function <System.Web.Services.Protocols.SoapMethodAttribute(
            "http://tempuri.org/MakeUpper",
            MessageStyle:=System.Web.Services.Protocols.SoapMessageStyle.
            ParametersInDocument)> MakeUpper(ByVal Data As String) As String
            Dim results() As Object = Me.Invoke("MakeUpper", _
        New Object() {Data})
            Return CType(results(0),String)
        End Function

        Public Function BeginMakeUpper(ByVal Data As String,
            ByVal callback As System.AsyncCallback, ByVal asyncState As Object)
            As System.IAsyncResult
            Return Me.BeginInvoke("MakeUpper", _
        New Object() {Data}, callback, asyncState)
        End Function
```

Listing 18.8 continued

```
Public Function EndMakeUpper(ByVal asyncResult As _
System.IAsyncResult) As String
    Dim results() As Object = Me.EndInvoke(asyncResult)
    Return CType(results(0),String)
End Function
End Class
End Namespace
```

Listing 18.8 shows a somewhat complicated class that imports a Web Service and XML class libraries and then creates a class that inherits from one of the base classes that understands the SOAP protocol. Notice too that this class implements both synchronous and asynchronous versions of the methods described in the WSDL document. This is the proxy class that you can use in your Web forms and components.

While this class may look a bit complex, there is good news. You do not have to code it yourself. The ASP.NET platform ships with a command line utility called WSDL.EXE that can automatically read a WSDL contract document and then produce the proxy class source code like you see in Listing 18.8. The following single-line command that will produce the proxy class you see in Listing 18.8.

```
wsdl http://localhost/aspx4devs/038x18/wsStrings.asmx /l:vb
➥/n:aspx4devs /o:wsStringsProxy.vb
```

TIP

If you are using Visual Studio.NET to create your SOAP clients, you will not need to use the WSDL utility at all. Visual Studio.NET will generate the SOAP proxy class automatically when you add a Web reference of the Web Service to your project.

The WSDL.EXE utility has a number of command line options. The table below shows the list of options, their arguments, and some usage comments.

Table 18.1 WSDL.EXE Options

Option	Example	Comments
/nologo	/nologo	Suppress the Microsoft banner.
/l	/l:vb	Indicate the language to use when generating the proxy class.
/server	/server	Generate an abstract server class instead of the usual client proxy.
/n	/n:myNameSpace	Indicate the namespace to use when generating the proxy class.
/o	/o:myProxy.vb	Indicate the name and folder to use as the output file.

Table 18.1 continued

Option	Example	Comments
/protocol	/protocol:SOAP	Indicate the default protocol to use when generating the proxy class. Options includeHttpGet,HttpPost, orSOAP.
/pu	/pu:myUserName	Indicate the username to use when calling into a secured Web Service.
/pp	/pp:myPassword	Indicate the password to use when calling into a secured Web Service.
/pd	/pd:myDomain	Indicate the domain name to use when calling into a secured Web Service.

NOTE

If you use WSDL.EXE with just the URL of the ASMX file that holds the Web Service, WSDL.EXE will generate a SOAP client proxy class using the C# language without any namespace with the filename that matches the name of the Web Service you are calling.

Once you generate the proxy class file, you need to compile it and then place it in the BIN folder of the virtual directory of the Web application that is calling the Web Service. This is an important point. The compiled proxy class goes in the *calling* Web, not the Web Service Web. The proxy will be used by the remote Web form or component to call into the distant Web Service.

Following is the single line command needed to compile the Visual Basic proxy class into a DLL assembly.

```
vbc /t:library wsStringsProxy.vb
```

Once you have the compiled DLL, place it in the BIN folder of a Web that you will use to host a Web form that will use the proxy class to call the Web Service.

Coding an ASP.NET Client Using a SOAP Proxy

Now that you have a compiled SOAP proxy class, you're ready to code a simple ASP.NET Web form client that uses that class. Using a SOAP proxy class is just like using any ASP.NET class library. All you need to do is add an import directive at the top of the page to include the namespace of the proxy class then code against the class as usual.

Listing 18.9 shows a completed ASP.NET Web Form that uses the SOAP proxy class shown in Listing 18.8.

Listing 18.9 WSSTRINGSCLIENT.ASPX—A Sample ASP.NET Web Form That Calls a Web Service

```
<%@ page description="wsStrings SOAP client" %>
<%@ import namespace="aspx4devs" %>

<script language="vb" runat="server">

sub MakeUpper_click(sender as object, args as EventArgs)

    dim oStrings as wsStrings = new wsStrings()
    returnValue.Text = oStrings.MakeUpper(dataInput.Text)
    oStrings = Nothing

end sub
</script>

<html>
<body>

<h2>wsStrings SOAP Proxy Client</h2>
<hr />

<form runat="server">
    Enter a lowercase string:
    <asp:Textbox id="dataInput" runat="server" />
    <asp:Button id="MakeUpper" Text="MakeUpper"
        OnClick="MakeUpper_click" runat="server" />
</form>

The SOAP return value is: <asp:Label id="returnValue" runat="server" />

</body>
</html>
```

In reviewing the server-side script block in Listing 18.9, you should notice that calling the Web Service through the proxy class looks no different than using any other .NET component. There is no evidence of XML, SOAP messages, or any other information kept in the WSDL document. That's because the SOAP proxy class handles all the work of dealing with the WSDL document and emitting & parsing incoming XML data.

Figure 18.2 shows an example of the ASP.NET Web Forms in a browser. Notice that the return value now appears with the Web page itself without any evidence of the XML document that appeared in the HTTP versions you saw earlier.

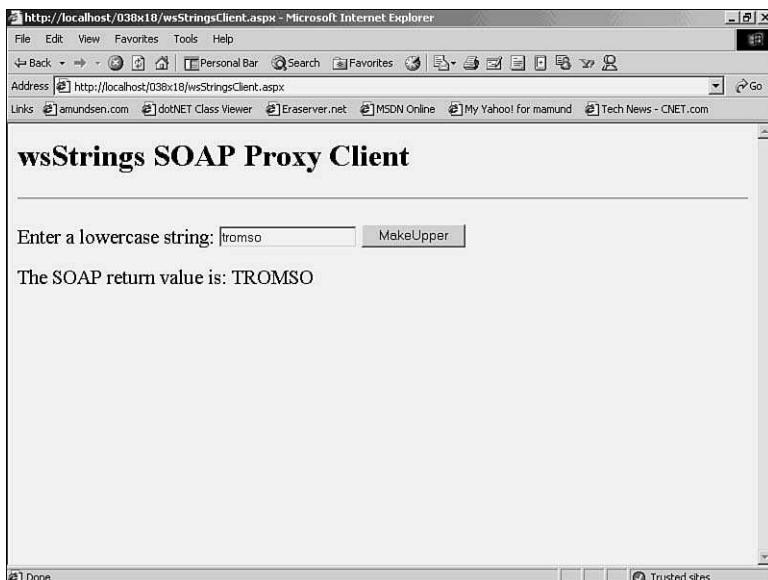


Figure 18.2

Running the ASP.NET Web form that calls a Web Service.

Summary

In this chapter, you learned how to interpret the WSDL document used to manage Web Service communication between two machines. You also learned how to create simple HTTP-GET and HTTP-POST test clients to call Web Services.

You can use the WSDL.EXE utility to generate a SOAP proxy client for use in your ASP.NET Web Forms or compiled components. This allows you to make Web Service calls using a simple .NET class and accept the return values to use directly in your Web forms or components as you would any other value.

PART VI

CONFIGURING AND DEPLOYING ASP.NET SOLUTIONS

- 19 Tracing, Debugging, and Optimizing Your ASP.NET Applications
- 20 Implementing ASP.NET Security
- 21 Configuring and Deploying Your ASP.NET Solutions

CHAPTER 19

Tracing, Debugging, and Optimizing Your ASP.NET Applications

In this chapter, you'll learn how to trace and debug your ASP.NET applications. You'll also learn how you can implement output and data-caching services to help optimize your ASP.NET solutions. Tracing services let you easily expose the contents of program variables, inspect program flow, and view data passed between the Web server and client browser. This makes it much easier to locate and fix problem sections in your code.

ASP.NET also ships with a standalone debugger that you can use to inspect running instances of your Web solution. This means you can load ASPX and compiled DLL source code into the debugger, mark breakpoints, and step through your Web application while inspecting memory locations, tracking program flow and more.

Finally, ASP.NET offers three types of caching services that can help you increase overall performance of your Web applications: *page caching*, which allows you to control how long individual pages are kept in memory before refreshing; *control caching*, which lets you implement partial page caching for memory-intensive sections of a single page; and *data caching*, which allows you to store just about any data object in memory on the Web server to reduce the resources needed to populate data-bound pages in your solution.

Tracing Services for ASP.NET

Tracing services allow programmers to track the value of variables and the execution of selected sections of code, and also view the contents of the HTTP header variables passed between the browser client and Web server. Often, this simple tracking and viewing of memory variables can expose errors in the code and make it easy for Web developers to locate and fix these bugs. In the past, ASP developers who wanted to trace the progress of their server-side scripts had to resort to placing `response.write` statements throughout the questionable page in order to determine the source of an error or to verify that certain code blocks were executing as expected. This was not only a time-consuming job, but also not always reliable.

With the release of ASP.NET, Web developers have a built-in tracing service that they can use to easily display HTTP header data and memory variable contents, and use them to validate code block execution. Placing the new Trace-class keywords `Trace.Write` and `Trace.Warn` directly into code blocks can do all of this. In addition, programmers can include conditional code that will produce trace results only under selected conditions. Finally, all trace information can be written to memory and stored for later review instead of being displayed directly in the executing page. This allows programmers to create trace logs at runtime without disrupting the display of the Web forms. These logs can then be inspected at a later time.

Page-Level Tracing

Adding trace service to an ASP.NET Web form is very easy. All you need to do is add a new attribute to the `Page` directive at the top of a Web form. For example, if you wanted to turn on tracing service for a Web form, you would add the following directive at the top of the page:

```
<%@ Page trace="true" %>
```

Now, when you run the page, you'll automatically see quite a bit of information, including a list of all the HTTP header variables passed between the client browser and the Web server. This default trace output covers a number of important areas:

- Request Details: This includes the Session ID, request and response encoding, request type, time of the request, and the response status.
- Trace Information: This shows the execution of the page class for the Web form. All the major events are listed along with the time (in milliseconds) it took to complete the code.
- Control Tree: This lists all the control objects that were added to the page, including displaying the parent/child relationship between all the control objects.
- Cookies Collection: This lists all the cookies currently passed in the request stream, along with their contents.
- Headers Collection: This lists all the header variables passed between client browser and Web server.
- Server Variables: This lists all the variables supplied and managed by the server and included in the response to the browser.

Figure 19.1 shows a sample of the default output of the trace service. As you can see, quite a bit of valuable information is available without adding any code to your page at all.

Request Details			
Session Id:	wiwqjj55ckffg1frhz0hn445	Request Type:	GET
Time of Request:	9/21/2001 12:52:03 PM	Status Code:	200
Request Encoding:	Unicode (UTF-8)	Response Encoding:	Unicode (UTF-8)

Trace Information			
Category	Message	From First(s)	From Last(s)
aspx.page	Begin Init		
aspx.page	End Init	0.000064	0.000064
aspx.page	Begin PreRender	0.000651	0.000588
aspx.page	End PreRender	0.000714	0.000062
aspx.page	Begin SaveViewState	0.001190	0.000477
aspx.page	End SaveViewState	0.001423	0.000233
aspx.page	Begin Render	0.001466	0.000043
aspx.page	End Render	0.002011	0.000544

Control Tree			
Control Id	Type	Render Size Bytes (including children)	Viewstate Size Bytes (excluding children)
_PAGE	ASP.tracing_aspx	1319	24
ctrl1	System.Web.UI.LiteralControl	141	0
ctrl2	System.Web.UI.ResourceBasedLiteralControl	717	0
ctrl0	System.Web.UI.HtmlControls.HtmlForm	437	0
ctrl3	System.Web.UI.LiteralControl	6	0
myLabel	System.Web.UI.WebControls.Label	67	88
ctrl4	System.Web.UI.LiteralControl	6	0
myButton	System.Web.UI.WebControls.Button	68	0
ctrl5	System.Web.UI.HtmlControls.HtmlForm	0	0

Figure 19.1

Viewing the default output of the ASP.NET trace service.

You can also customize the trace output using the `Trace.Write` and `Trace.Warn` methods. These allow you to add code to your page that will write entries directly to the trace output for viewing on the screen.

For example, the code in Listing 19.1 shows the use of both the `Trace.Write` and `Trace.Warn` methods to track the execution of code blocks. The `Trace.Write` method is used to mark the start and end of the `Page_Load` event. The `Trace.Warn` method is used to mark both the execution of the inner if clause in the `Page_Load` event and in the click event of the page button.

Listing 19.1 TRACING.ASPX—Customizing Trace Output with Trace.Write and Trace.Warn Methods

```
<%@ Page description="tracing demo" trace="true" %>

<script language="vb" runat="server">

Sub Page_Load(Src as Object, E as EventArgs)
    Trace.Write("Page_Load Starts", "ASPx4DEVS")
    If E.PostBackUrl = "" Then
        If E.PostBackValue = "Click" Then
            Trace.Warn("Page_Load Ends", "ASPx4DEVS")
        End If
    End If
End Sub


```

Listing 19.1 continued

```
If Not Page.IsPostBack Then
    myLabel.Text = "Welcome to <b>ASP.NET for Developers</b>."
    Trace.Warn("Executing for the First Time!", "ASPXDEVS")
End If
Trace.Write("Page_Load Ends", "ASPX4DEVS")
End Sub

sub myButton_Click(Src as object, E as EventArgs)
    Trace.Warn("Executing myButton_Click!", "ASPXDEVS")
end sub

</script>

<html>
<body>

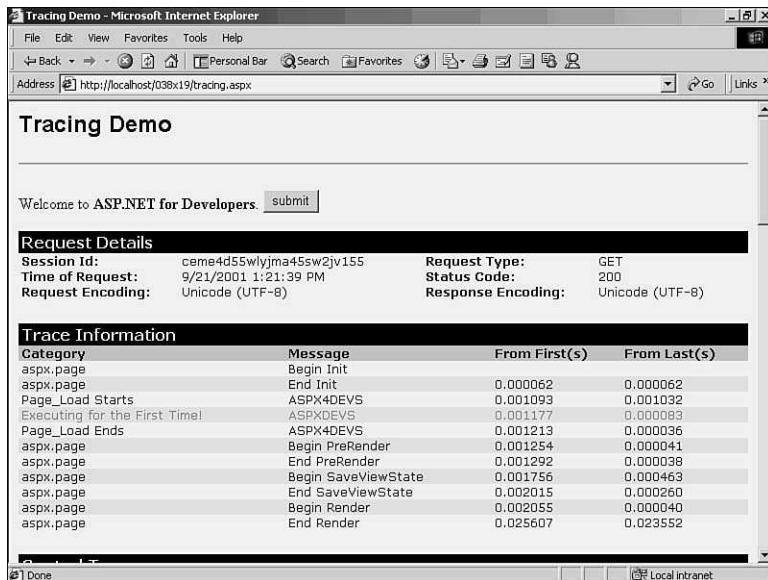
<h2>Tracing Demo</h2>
<hr />

<form runat="server">
    <asp:label id="myLabel" runat="server" />
    <asp:button id="myButton" text="submit" OnClick="myButton_Click"
        runat="server" />
</form>

</body>
</html>
```

You can see from the code in Listing 19.1 that both the `Write` and `Warn` methods of the `Trace` object accept two arguments. The first is the string you wish to appear in the trace output. The second argument is the category under which you'd like the message to appear. This allows you to send several messages into the trace output and later sort them by category.

All messages produced by the `Write` and `Warn` methods appear in the Trace Information section of the trace output. Figure 19.2 shows an example trace log from the code shown in Listing 19.1. (When you view a trace log on your own screen, you'll notice that the warning messages appear in red, while the other messages appear in standard black type.)

**Figure 19.2**

Viewing customized trace output.

Application-Level Tracing

While it is handy to be able to display trace outputs for your pages, sometimes you will want to create the trace logs and save them to a file for later viewing. Doing so allows you to execute a series of pages and test for the conditions of cookies, session variables, and other values that exist across pages. This also allows you to produce trace logs without disrupting the display of your pages at runtime.

The ASP.NET trace service has such a feature. All you need to do is modify the WEB.CONFIG file of your application to turn on application-level tracing. You can then remove the `trace="true"` attributes on the individual pages. When this is done, all trace data will be sent to memory where it can be accessed later.

Following is the entry you need to add to your WEB.CONFIG file. This activates the application-level trace service, suppresses output for each page, and stores up to ten page views in memory at one time.

```
<configuration>
  <system.web>
    <trace enabled="true" requestLimit="10" pageOutput="false" />
  </system.web>
</configuration>
```

After adding this to your WEB.CONFIG file and removing the `trace="true"` from your pages, you can execute these pages in your browser and the trace service will write the output to a log for you to view later.

NOTE

If you wish to exclude some pages from application-level tracing, you can add `trace="false"` to the page. This will tell the trace service to skip this page when creating application trace logs.

For example, if application-level tracing is turned on for the application that hosts the page shown in Listing 19.1, you can load the page and press the button several times. This will place traces into the in-memory trace log. When you run the pages, you'll see just the regular display of the Web forms even though the traces are added to the in-memory log.

To view the in-memory trace log, you need to point your browser to a special URL within the application. This URL will then call up the trace logs in a list for your viewing. To view the trace logs, you need to enter the following request in your browser:

`http://<servername>/<applicationname>/trace.axd`

For example, if the page from Listing 19.1 were on the LOCALHOST server in an application called 038x19, then the complete URL would be:

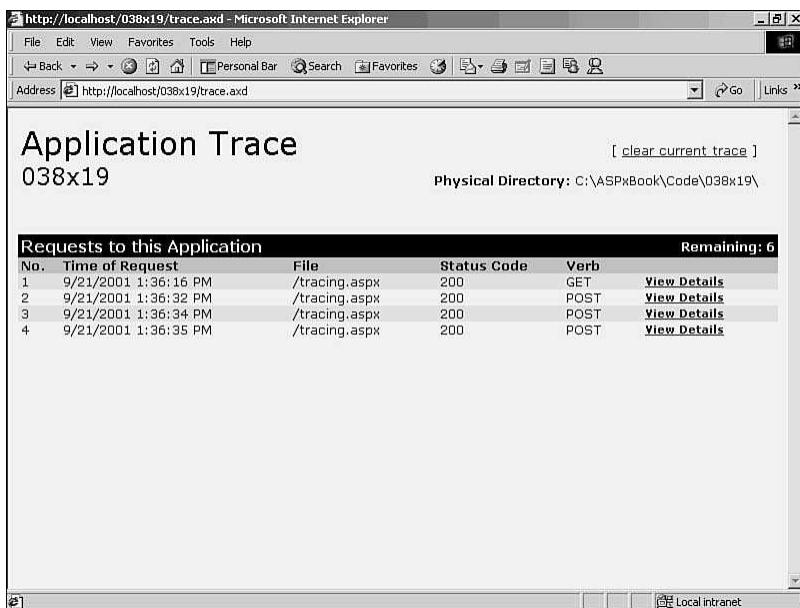
`http://localhost/038x19/trace.axd`

Figure 19.3 shows the trace log listing displayed when you point your browser to this URL. You can see the list of all the traces placed into memory. You can view the contents of each trace by clicking on the “View Details” link on the page.

NOTE

The TRACE.AXD URL does not actually point to a real document. When you send this request to the ASP.NET Web server, a special HTTP handler (the trace handler) intercepts the request and then produces the log listing for you. HTTP handlers are a very powerful, advanced feature of ASP.NET, and are beyond the scope of this book.

Along with viewing the saved trace logs, you can clear them out of memory using the hyperlink at the top of the TRACE.AXD page. It is also important to note that trace logs stay in memory until the application is restarted or the Web service, IIS, or the machine itself is restarted. Once any of these events occur, the traces are removed from memory.

**Figure 19.3**

Viewing the *in-memory* trace log.

Using the Standalone Debugger

While the new ASP.NET trace service is valuable, sometimes you need a more powerful way to track and monitor the execution of your Web applications. For this you need a more full-featured debugging service that allows you to do such things as load source code, set breakpoints, or inspect and even modify memory locations at runtime. The ASP.NET install package includes just such a tool: the CLR Debugger, a standalone debugging application that you can use for both desktop and Web-based solutions. In this section, you learn how the debugger works and how you can use it to debug your ASP.NET solutions.

A Quick Tour of the Standalone Debugger

Before getting into the details of using the debugger with your Web applications, it is important to spend a little time getting to know the CLR Debugger and how it works. In this section, you'll learn how to locate and launch the debugger and learn a few of the details of how it works and how you can navigate through the application.

First, the CLR Debugger is installed when you install the .NET framework itself. It is not a special part of ASP.NET at all; it's part of the .NET framework. This means that the CLR Debugger can be used for both desktop and Web solutions. The CLR Debugger executable is called `DBGCLR.EXE` and, by default, is installed at the following location on your workstation:

`C:\Program Files\Microsoft .NET\FrameworkSDK\GuiDebug\DbgCLR.exe`

If you did not install the .NET runtimes in the default location, you may need to search a bit before you find this program. Once you find it, you can double-click on the EXE file to launch the program. Figure 19.4 shows the CLR Debugger loaded and running.

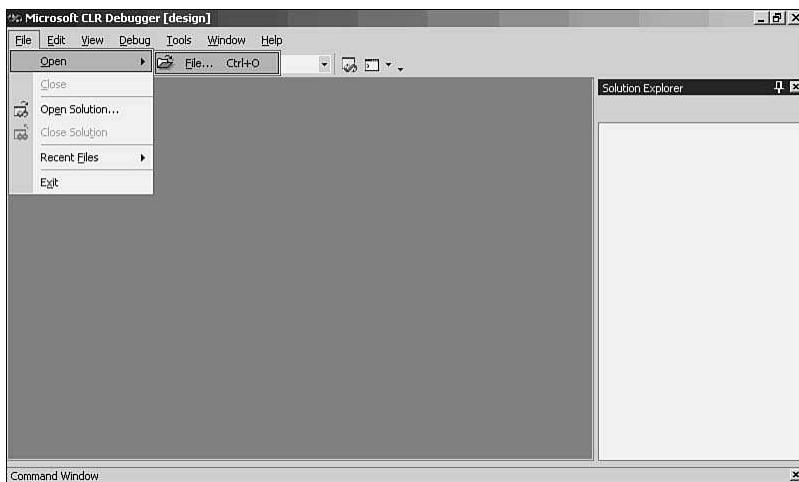


Figure 19.4

Viewing the CLR Debugger.

When you first load the debugger, not much is viewable. In order to activate most of the menus, you need to load a source code page. For example, Listing 19.2 shows the source code of a very simple Visual Basic .NET component. Figure 19.5 shows this document loaded into the debugger using the File Open menu option.

Listing 19.2 SAMPLE.VB—Source Code for a Simple VB .NET Component

namespace ASPX4Devs

```
public class testDebugging

    public function showString(byval data as string) as string

        dim sTemp as string

        sTemp=Data & "<br />" & Data
        sTemp = sTemp.ToUpper()

        return(sTemp)

    end function

end class

end namespace
```

Most of the options you will use are found within the Debug main menu option. This is where you'll start the debugging process, and it's also where you'll find a number of additional windows that can be displayed while you are running the debugger against your Web application.

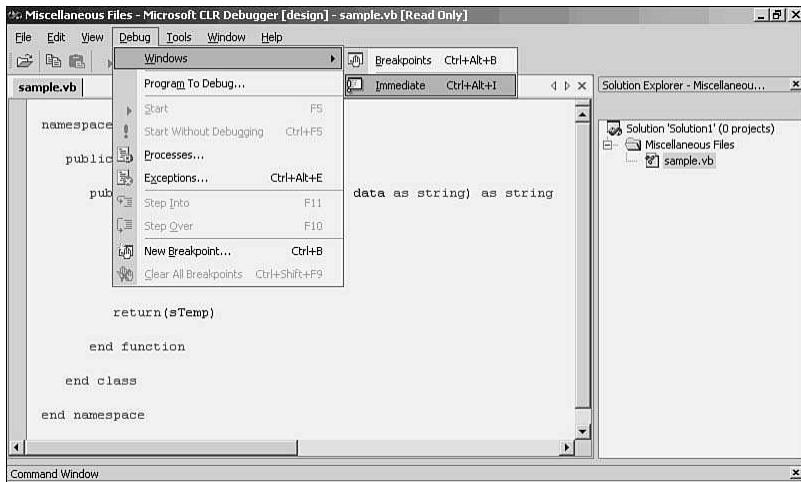


Figure 19.5

Viewing the component source code in the debugger.

In the next section, you'll learn how you can perform live debugging on an existing ASP.NET Web form.

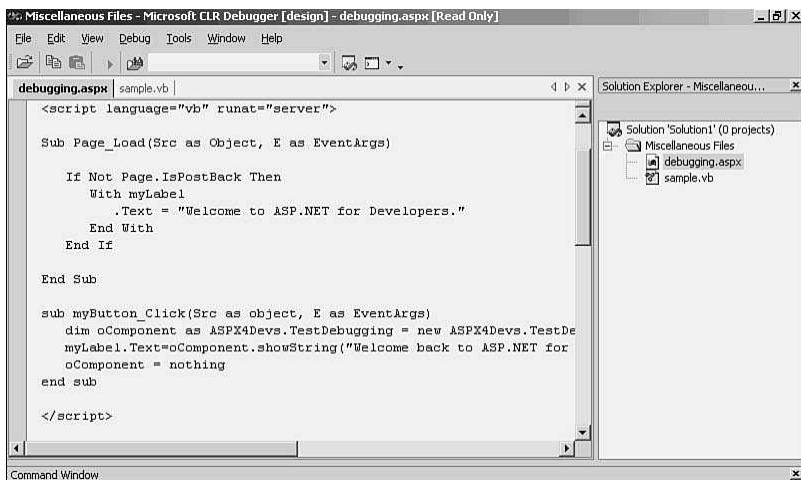
Debugging ASP.NET Pages

In order to perform live debugging on a running Web form, you need to perform a few tasks that set the stage. First, you need to load the source code for your ASPX page(s) into the CLR Debugger. Next, you need to launch an instance of the Web application in your browser. Finally, you need to attach to that running instance within the debugger itself. Once these three tasks are done, you're ready to start debugging your application!

Before you can debug your ASPX pages, you need to include the `debug="true"` attribute on the page directive. This tells the runtime compiler to create debugging information when the page is first compiled. Following is a sample page directive that shows how this would look in your ASPX page:

```
<%@ Page debug="true"%>
```

Next, you need to launch the debugger and load one or more ASPX pages into the debugger. For example, Figure 19.6 shows the debugger loaded with an ASPX page called DEBUGGING.ASPX. This page not only has typical ASP.NET code, but it also calls a compiled component. You'll learn how to debug compiled components later in this chapter.

**Figure 19.6**

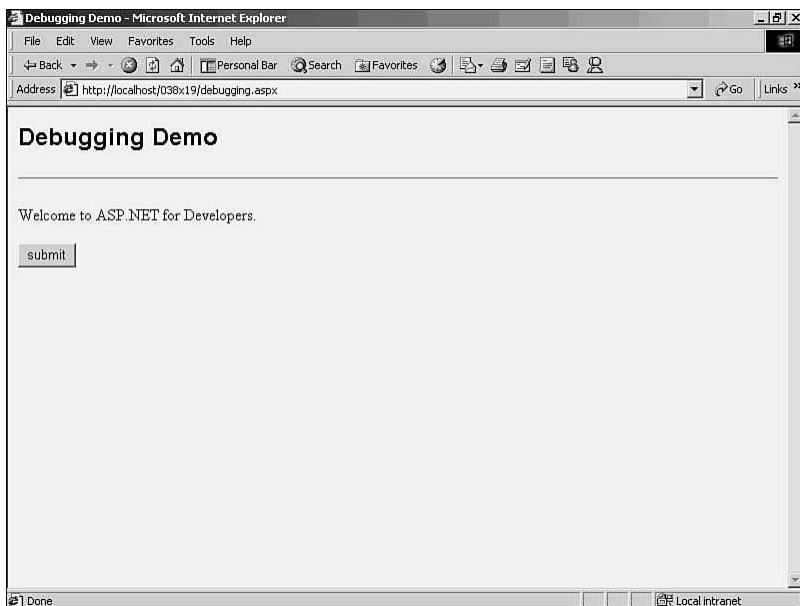
Loading the DEBUGGING.ASPX file in the CLR Debugger.

Next, you need to load the same page in your browser. This ensures that a running instance of the ASP.NET application is in memory and available for debugging. For example, if the DEBUGGING.ASPX page were part of the 038x19 application on your LOCALHOST machine, you'd type the following URL into your browser:

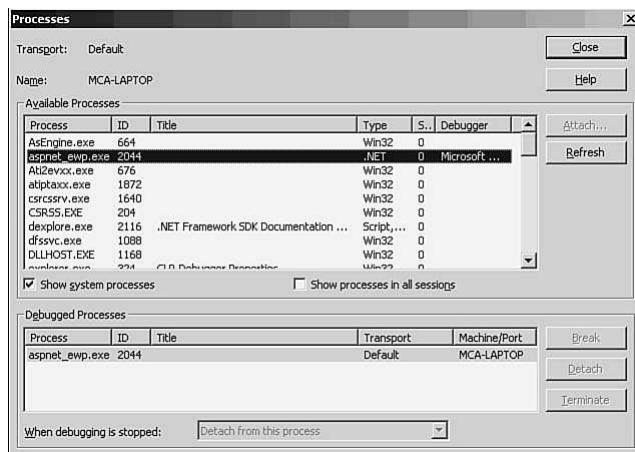
<http://localhost/038x19/debugging.aspx>

Figure 19.7 shows this running instance in the Microsoft Internet Explorer browser.

Next, you need to return to the CLR Debugger and attach to the running process that was started when you loaded the page into your browser. To do this, switch to the CLR Debugger and select Debug, Processes... from the main menu. This will call up the Processes dialog box. Be sure to check the "Show system processes" checkbox at the bottom left of the dialog. Then locate and double-click the aspnet_ewp.exe entry in the list. This is the executable that handles all ASP.NET Web applications on the machine (see Figure 19.8). Once you have attached to the aspnet_ewp.exe process, you can press the Close button and return to the main dialog for the CLR Debugger.

**Figure 19.7**

Viewing the running instance of DEBUGGING.ASPX.

**Figure 19.8**

Attaching to the ASPNET_EWP.EXE process.

At this point, your debugger is active and you can start setting breakpoints in your ASP.NET Web form. For example, you can now click in the leftmost margin of any executable line in the debugger to set a breakpoint somewhere in the Page_Load event. Now, when you switch to the browser and press the Refresh button, it will force the page code to re-execute and the debugger will halt execution on the highlighted line (see Figure 19.9).

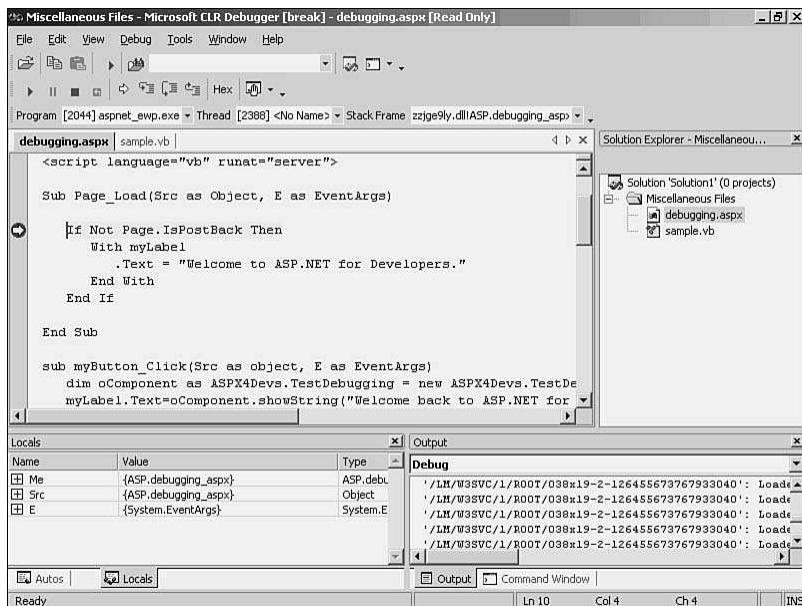


Figure 19.9

Encountering a breakpoint in the debugger.

At this point, you can step through the code using the F11 key or by selecting Debug, Step Into from the main menu.

You can also view several other windows that give you additional information about the running code. Select Debug, Windows from the main menu to see a list of possible additional windows. Some of the more interesting ones are listed here.

- **Breakpoints:** This window lists all the breakpoints you have set in all the documents loaded into the debugger.
- **Watch 1-4:** These windows allow you to drag objects and variables from the code and drop them into the watch windows. You can then trace the values loaded into the objects as the program executes.
- **Locals:** This window shows the objects and variables that are local to the running process. The contents of this window change as you move from one method to the next.

- Call Stack: This window lists all the classes and methods that have been called.
- Modules: This window shows all the physical assembly modules loaded into memory.
- Disassembly: This window shows the actual MSIL intermediate code that is stored in the compiled files. (In some cases, this will not be available at runtime.)

Debugging Compiled Components

You can also debug compiled components using the CLR Debugger. To do this, you need to compile your components with the debug+ command-line option. This instructs the compiler to create the proper debug information file and store it in the same location as your DLL. This file is given the same name as your compiled component, but has a file extension of PDB (program debugging) instead of DLL. Following is a command line compile command for a simple component that includes the debug+ option:

```
vbc /target:library /debug+ sample.vb
```

Once the component is properly compiled, you can load the source code into the debugger and set breakpoints for it just as you would any ASPX page you are debugging.

Figure 19.10 shows the debugger in breakpoint mode on a line within the compiled component. Note also in the bottom left of the screen the display of the local variables showing the status of the various string variables declared within the method.

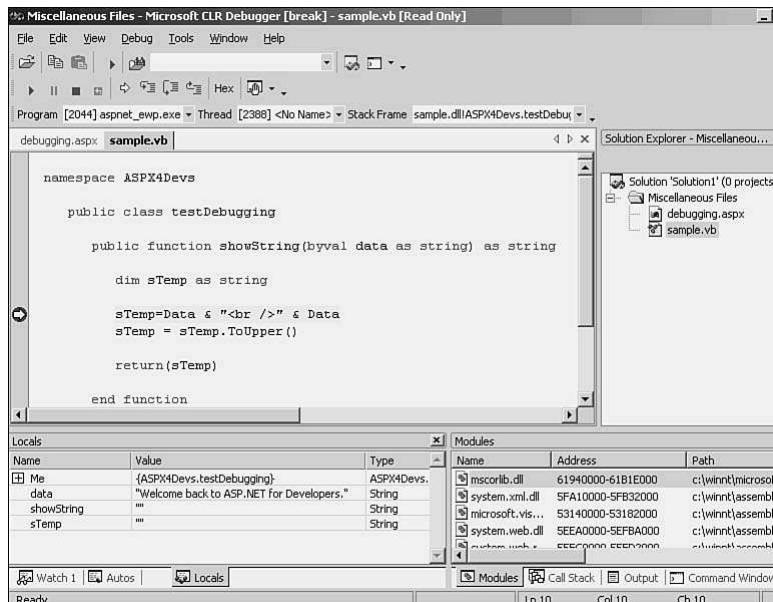


Figure 19.10

Viewing a breakpoint within a compiled component.

You can even modify memory variables while debugging. For example, while the debugger is stopped in the component, you can alter the contents of the return value. This allows you to fix certain execution bugs while the code is running in order to verify the results before recompiling.

For example, Figure 19.11 shows the `sTemp` return value was altered by adding "This is new text
" to the string. This can be done by double-clicking on the string value in the Locals window and entering the new text. Then, when the program is resumed, the new string value will be displayed in the browser.

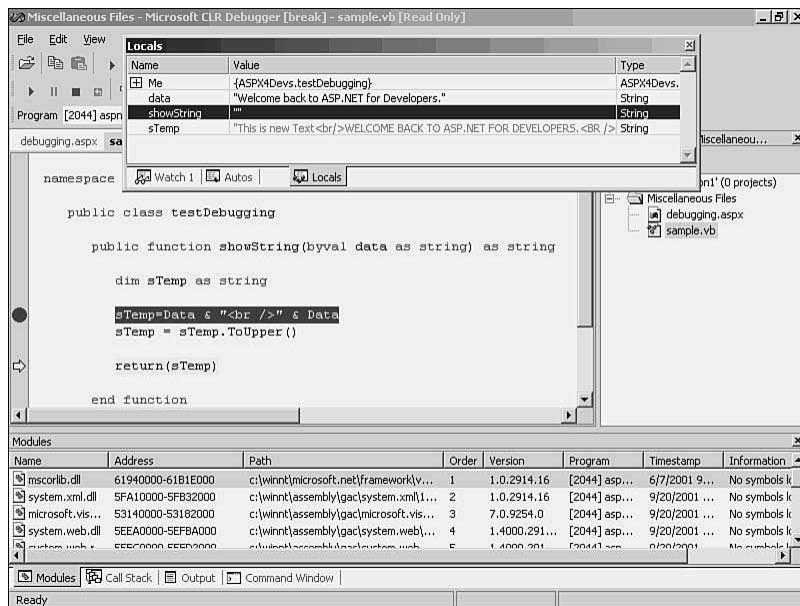


Figure 19.11

Runtime debugging an ASP.NET application.

Optimizing Your ASP.NET Applications with Caching Services

Once you have your applications up and running, you'll probably want to optimize performance as much as you can. One of the easiest ways to do this is to instruct the Web server to hold commonly requested pages in memory instead of executing them from code every time. You can also increase performance by reducing the number of times you make calls to a remote database or file store by storing the complex data in memory on the Web server. In both cases, this process of storing information in RAM memory for later recall is called *caching*.

In the past, caching services for ASP solutions were only available using third-party products or home-built solutions. However, ASP.NET has caching services built into the product. There are actually two different caching systems that are available with ASP.NET. *Output caching* allows you to store one or more pages, or partial pages, in memory for later recall. A second type of caching, called *data caching*, lets you store any kind of string or complex object in memory for later recall. This is a great way to store commonly used data tables or data files.

In this section, you'll learn how to use both output and data caching in your ASP.NET solutions.

Implementing Page Output Caching

The easiest form of caching you can implement is page-output caching. This allows you to instruct the runtime to store an entire page in memory for later use. When this is done, each call for the page is served from RAM memory instead of executing the actual code. This can add a great deal of speed to pages that call data from databases or pull images from disk or from somewhere else on the Internet.

To implement page output caching, you need to add a new directive at the top of the page. This directive tells the ASP.NET runtime how long to hold the page in memory before refreshing it from disk, and whether to watch the query string (or some other variation on the page call in determining how to cache) and recall the page. Following is a typical output cache directive:

```
<%@ OutputCache Duration="300" Location="any" VaryByParam="*" %>
```

In the preceding line, the instructions are to hold the page in memory for 300 seconds (five minutes). After five minutes, the page will be rendered again from the code base. This will allow minor changes to be collected and displayed on a regular basis. The *Location* attribute indicates where the page should be cached. Valid values here are “Client” (the Web browser), “Server” (the Web server), “Downstream” (some other server downstream from the Web server), “Any,” and “None.” The default is “Any.” The *VaryByParam* attribute is set to treat each set of query string parameters as a unique page. This makes sure that as users pass various values to a page, they don’t receive the same cached page each time. There is also an attribute called *VaryByHeader* so that different devices (mobile phones, 3.2 browsers, and so on) can have their own custom-cached pages.

NOTE

There is also a *VaryByCustom* setting that allows you to create your own caching parameters. This setting also requires that you build your own compiled code to handle the caching details. Check the .NET documents for details on how to do this.

For example, Listing 19.3 shows a page that reports the time. However, since the output caching is set to hold the page in memory for ten seconds, repeated attempts to refresh the page will result in the time display remaining constant until ten seconds have passed.

Listing 19.3 CACHEPAGE.ASPX—A Simple Example of Page-Output Caching

```
<%@ Page description="page caching demo" %>
<%@ OutputCache Duration="10" Location="any" VaryByParam="none" %>

<script language="vb" runat="server">

Sub Page_Load(Src as Object, E as EventArgs)
    myLabel.Text = DateTime.Now().ToString()
End Sub

</script>

<html>
<body>

<h2>Page Caching Demo</h2>
<hr />

<form runat="server">
    <p>
        <asp:label id="myLabel" runat="server" />
    </p>
</form>

</body>
</html>
```

Implementing User Control Output Caching

Page-output caching is handy when you want to store the entire page in memory and not refresh any part of it for a while. However, in some cases, you may just want to cache certain portions of a page and allow the rest to be dynamically built each time. For example, you might have a menu control on your home page that makes repeated calls to a remote location to render large image files. While the rest of the page needs to be dynamic, you'd like the image portion of the page to be cached in memory to speed performance.

With ASP.NET, you can achieve this effect by creating a user control and then caching only the user control itself. For the example mentioned above, you can create a user control that renders the images and then cache that in memory. Listing 19.4 shows a user control that is set to store itself in the cache and refresh every ten seconds.

Listing 19.4 UCTEMP.ASCX—User Control That Uses Caching

```
<%@ control classname="ucTemp" %>
<%@ outputcache duration="10" varybyparam="none" %>

<script language="vb" runat="server">

sub Page_Load(Src as object, E as EventArgs)
    myLabel.Text=DateTime.Now()
end sub

</script>

<asp:Label id="myLabel" runat="server" />
```

Now you can create a page that uses this custom control. Listing 19.5 contains code that uses the custom control from Listing 19.4. Notice that this page has no caching directive—it is a dynamically built page. When this page is executed in a browser, repeated refreshes will update the time on the main page, but update the time on the cached page only at ten-second intervals.

Listing 19.5 CACHECONTROL.ASPX—Calling a User Control That Is Cached

```
<%@ Page description="user control caching demo" %>
<%@ register TagPrefix="aspx4devs" TagName="ucTemp" Src="ucTemp.ascx" %>

<script language="vb" runat="server">

Sub Page_Load(Src as Object, E as EventArgs)
    myLabel.Text = DateTime.Now().ToString()
End Sub

</script>

<html>
<body>

<h2>User Control Caching Demo</h2>
<hr />

<form runat="server">
    <p>
        In Page Code:<br />
        <asp:label id="myLabel" runat="server" />
    </p>

    <p>
        User Control:<br />
    </p>
```

Listing 19.5 continued

```
<asp:ucTemp id="myControl" runat="server" />
</p>
</form>

</body>
</html>
```

Utilizing Data Caching

Finally, in some cases you might not want to store an entire page or a partial page. Instead, you might want to store a dataset or some other complex object that can be quickly called from any page or control in the solution. In this case, what you want to utilize is *data caching*. This feature of ASP.NET lets you pull data from a disk file or database and then store it in memory on the server. This memory is available to all users and all pages within the application.

You implement data caching using the Cache object in code. Any object can be stored in the data cache. For example, you can store simple strings in the cache, or arrays, or even complex datasets. Using the Cache object is very similar to using the Session object. You simply create a newly named property in the cache and fill it with the data you wish to store. Now any page or user in the same application can access the contents of cache at any time.

Listing 19.6 shows a page that pulls a dataset from a SQL Server database. This data is then stored in the data cache. Now, when the page is refreshed, or even revisited within the same session, the data will be pulled from memory instead of from the database.

Listing 19.6 CACHEDATA.ASPX—Using the Data Cache to Hold Information from a Database

```
<%@ Page description="cache data demo" %>
<%@ import namespace="System.Data" %>
<%@ import namespace="System.Data.SqlClient" %>

<script language="vb" runat="server">

Sub Page_Load(Src as Object, E as EventArgs)

    Dim dv As DataView

    dv = Cache("aspx4devs")

    If dv Is Nothing

        Dim cn as SqlConnection = _
            new SqlConnection("server=localhost;database=pubs;uid=sa;password=")

        ' Create a new dataset
        Dim ds as New DataSet("MyDataset")
        ' Create a new adapter
        Dim da as New SqlDataAdapter("SELECT * FROM authors", cn)
        ' Fill the dataset
        da.Fill(ds)
        ' Cache the dataset
        Cache("aspx4devs") = ds
    End If
End Sub

```

Listing 19.6 continued

```
Dim da as SqlDataAdapter = _
    new SqlDataAdapter("select * from Authors", cn)

Dim ds As New DataSet
da.Fill(ds, "Authors")

dv = New DataView(ds.Tables("Authors"))
Cache("aspx4devs") = dv

statusMsg.Text = "Dataset created explicitly"
Else
    statusMsg.Text = "Dataset retrieved from cache"
End If

dbGrid.DataSource = dv
dbGrid.DataBind()

End Sub

</script>

<html>
<body>

<h2>Data Caching Demo</h2>
<hr />

<form runat="server">
    <p>
        <asp:Label id="statusMsg" runat="server" />
    </p>

    <p>
        <asp:DataGrid id="dbGrid" runat="server" />
    </p>
</form>

</body>
</html>
```

When this page is loaded in a browser the first time, the cache is checked. Since it is empty, the code makes a call to the database, pulls the required data and then stores it in the cache for future use. The second time this page is loaded, the information will be found in the cache and the call to the database server will be skipped.

Summary

In this chapter, you learned how tracing services let you easily expose the contents of program variables, inspect program flow, and view data passed between the Web server and client browser. Tracing services make locating and fixing problem sections in your code much easier. You also learned how you can use the CLR Debugger that ships with ASP.NET to load ASPX and compiled DLL source code, mark breakpoints, and step through your Web application while inspecting memory locations, tracking program flow, and more.

Finally, you learned how to optimize your applications using three types of caching services that can help you increase overall performance of your Web applications. This includes page caching, which allows you to control how long individual pages are kept in memory before refreshing; control caching, which lets you implement partial-page caching for memory-intensive sections of a single page; and data caching, which allows you to store just about any data object in memory on the Web server to reduce the resources needed to populate data-bound pages in your solution.

CHAPTER 20

Implementing ASP.NET Security

In this chapter, you'll learn how to secure your Web solutions using the powerful security features of ASP.NET. One such feature, Forms Authentication, allows you to create simple login forms and control access to all pages in your solution. You'll also learn how to implement a Windows security scheme requiring your users to sign in with their own Windows user names and passwords and use the control features of the Windows operating system to secure your solution.

Before delving into ASP.NET's security features, this chapter reviews important security concepts including the meaning and purpose of authentication, authorization, and impersonation. The way ASP.NET and Internet Information Server (IIS) interact also is covered.

When you have completed this chapter, you'll understand a number of important security concepts as well as know how to design your solution to support the various security engines that ship with ASP.NET.

Important Security Concepts

Before getting into the details of designing and implementing security schemes for your Web solutions, it is important to spend some time reviewing and clarifying a few key security concepts. The three key concepts that are important to understand when using the ASP.NET security models are

- Authentication: Identifying users that access your application.
- Authorization: Determining users' rights when they attempt to perform tasks in your solution.

- Impersonation: Establishing the Windows identity used to execute your application code.

In addition to these three key concepts, you also should understand the division of duties between the ASP.NET security system and the security features of Microsoft's IIS.

Authentication

Authentication is the process of determining the identity of the user visiting your Web pages. In other words, "who are you?" This is the first step to be performed in any security system. Before you can determine whether the user is allowed to read or edit content in your Web site, you must determine the identity of the user.

The most common way to perform authentication is to ask for a username/password pair of values. This value pair can then be compared to a data store that contains recognized users for your Web solution. Some systems that require only minimal security may ask for only a username, but this is usually not adequate for most scenarios.

The username/password pairs can be stored in a number of ways. First, they may be stored within the operating system itself. For example, you might want to compare the login information to the list of Windows users on the Web server or in the Windows domain. In some systems, the login list is stored in a database such as SQL Server 2000 or Microsoft Access. Also, username/password data might be stored in a disk file in an XML or comma-delimited format.

Finally, if the username/password list is very small, the data might be compiled directly into the application itself. This is often done for small management solutions that only need to validate the administrator account.

Wherever the information is stored, the process of authenticating users is the first line of defense in any security system.

Authorization

Authorization is the process of determining whether the authenticated user has sufficient rights to perform the requested task. In other words, "are you allowed to do that?" You can perform this security step any time users request a Web page, attempt to save data to the site, or even attempt to read data from a remote database.

The authorization process involves comparing the authenticated identity (for example, the user name), against a list of resources that the user might attempt to access. Typically the list will include both the resource and the list of users. Sometimes the list will also include multiple access levels that can be applied to each user for that resource. Common access levels are read, write, create, and delete. Different resources might require specialized access levels including update, administer, and so on.

There are a number of ways to implement authorization security in your ASP.NET applications. First, you can take advantage of the existing authorization features of the underlying operating system. This means you can use the Windows file and folder

security to test the logged-in user's rights against the access control lists (ACLs) applied to the files and folders on the server. This is usually the most reliable way to implement authorization services.

Another common authorization scheme involves storing a list of resources and access rights in a database or some other disk file. In this case, all attempts to access a resource would require checking the database to determine whether the current user has the proper access rights to perform the task. This process of storing authorization information in a data file is very common for Web solutions.

Probably the most difficult aspect of implementing a robust authorization scheme is capturing the user request for a secured resource in an efficient way. Typically, Web programmers will add code to the top of each server-scripted page to check the user identity against the authorization data store. However, in some cases this method is not very effective. For example, simple HTML pages that do not contain server script are difficult to secure. The same is true for non-text resources such as images, multimedia files, and binary download packages.

No matter what method is used, authorization is an essential part of most Web security schemes.

Impersonation

Impersonation is the act of executing a resource request using the identity of the logged-on user instead of the default identity of the Web application itself. This is a very important and often misunderstood concept. When IIS is first installed, it is configured to use the "local machine" account to access all Web resources. This account is usually called IUSR_<machinename> and has special rights to read all Web pages and execute all scripts on the local machine. It is also possible to create a special user account and apply that new account as the default local machine account for all Webs. You can even establish a unique user account for each Web application on the server.

However, establishing user accounts for the Web applications is not the same thing as implementing impersonation. Impersonation is the process of executing access requests using the identity of the user that logged-into the Web, not the user identity applied to the Web itself. In other words, you must be sure to set up your solution to assume the identity of the logged-in user when executing code.

The most common way to implement impersonation on Windows servers is to implement Windows authentication and then use the Windows operating system authorization service (ACLs) to control user access to the available files and folders. The last step in this process requires adding a setting in the Web configuration to force the server to assume the logged-on user identity whenever making a resource request. It is also possible to write code within your application to make a resource request using the logged-in user identity.

Impersonation is not often needed for most public Web solutions. However, it is particularly useful when you are implementing Web-based solutions for internal networks where highly sensitive data is stored on shared servers.

Interaction with IIS

The final key security issue to consider is the interaction between ASP.NET and IIS. Each plays an important role in the implementation of a successful security scheme for your Web solutions and it is important to know the roles and responsibilities of both ASP.NET and IIS when you implement security for your Web applications.

The Role of IIS

In all cases, requests for Web resources are first presented to IIS for handling. At this point, IIS can perform a number of possible security checks in an attempt to authenticate the user making the request. Figure 20.1 illustrates the security decisions that IIS makes.

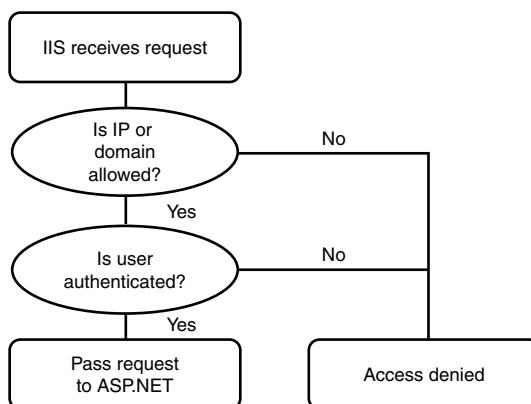


Figure 20.1

IIS security flow for Web requests.

First, IIS can check a list of IP addresses to determine if the requesting IP is allowed to access resources on the server. If not, an “access denied” message is sent to the requestor.

If the IP is allowed, IIS can then attempt to authenticate the requesting entity using any one of the common authentication methods including basic, digest, or Windows NTLM authentication. If the user credentials are not found in the proper data store (usually the operating system user list), the requestor is sent an “access denied” message. However, if the proper credentials are passed to IIS, the request is passed along to the ASP.NET platform for handling.

Once the resource request passes IIS, it is sent on to the ASP.NET platform for handling. What happens here depends on how your ASP.NET solution is configured.

The Role of the ASP.NET with Windows Authentication

If your Web solution is configured to support Windows authentication, then the ASP.NET platform will assume a known Windows identity and execute the code accordingly.

First, ASP.NET will check to see if impersonation is turned on (see Figure 20.2). If it is, then ASP.NET will assume the identity of the user authenticated by IIS. If impersonation is not enabled, then ASP.NET will assume the identity of the default local machine account for this particular Web solution.

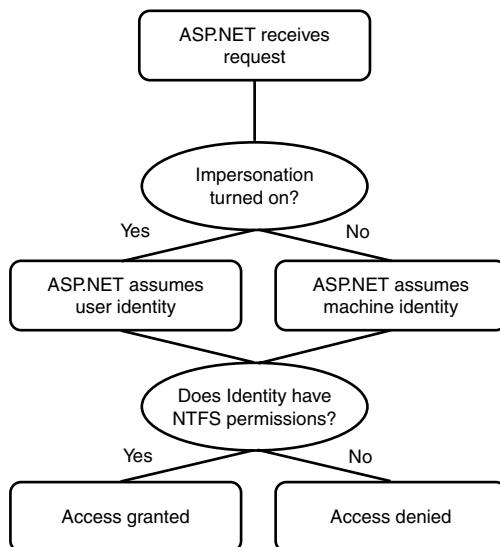


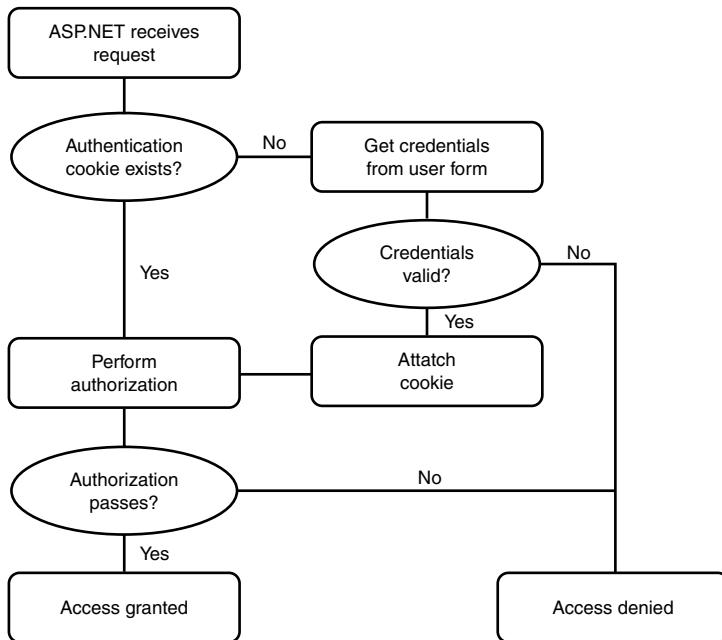
Figure 20.2

ASP.NET security flow for Windows authenticated requests.

Next, ASP.NET uses the assumed identity (user or machine) and checks this identity against the NTFS access control list applied to the requested resource (file or folder). If the identity has access to the resource, the code is executed and the page is sent back to the requesting party. If the identity does not have rights to the resource, an “access denied” message is sent instead.

The Role of ASP.NET with Forms Authentication

If your Web solution is configured to support forms authentication, then the ASP.NET platform will deliver a user login page and start the process of completing both the authentication and authorization processes, as shown in Figure 20.3.

**Figure 20.3**

ASP.NET security flow for forms authenticated requests.

First, ASP.NET checks to see if an authentication cookie was passed from the requesting client. If not, then the user is presented a custom user login form. Once the user enters the requested credentials (usually username and password), ASP.NET checks the credentials against the data store of user names and passwords. If the credentials fail, the user will be presented with a “denied access” message. If the credentials are correct, ASP.NET will attach the authentication cookie to the request and pass it along.

Next, ASP.NET will use the authentication cookie to attempt to authorize the user for the request. This can be done using some form of data store either accessed by ASP.NET directly or through code in the Web application. Once the authorization passes, the page is sent to the user. If not, the user receives a “denied access” message instead.

Now that you have a good understanding of how user’s requests for Web resources are handled by IIS and ASP.NET, you’re ready to build example ASP.NET solutions that illustrate these concepts.

Implementing Windows-Based Security

The simplest security model to implement for your ASP.NET applications is Windows-based Security (WBS). This method relies heavily on the Windows operating system as well as settings in the IIS. In fact, you can successfully implement WBS without making any changes to your ASP.NET pages themselves.

Another advantage of WBS is that it allows you to easily secure non-ASP resources such as images, multi-media files and other binary formats. Finally, the WBS scheme allows you to take full advantage of the power of Windows usernames & passwords as well as built-in ACLs applied to files and folders.

While there are a number of advantages to using Windows-based security, there are also some drawbacks. First, to work best, WBS requires that you define a new Windows user for each login account on the server. This can get tedious if you have a very large user base or a user base that changes membership frequently. Second, you cannot use WBS to secure virtual locations such as custom ISAPI extensions or ASP.NET handlers. Finally, WBS is tied directly to the Windows operating system. This means that you probably should not use WBS if you ever hope to run your .NET solution on some other operating system in the future.

Windows-Based Authentication

The first step in implementing WBS is establishing the authentication process. Making entries in the Web applications configuration file easily does this. For example, create a new Web application on an ASP.NET server in a folder called WBS. Next create a WEB.CONFIG file and place the following code in the configuration file:

```
<configuration>
  <system.web>
    <authentication mode="Windows" />
    <identity impersonate="false" />
  </system.web>
</configuration>
```

You'll notice the `<system.web>` section contains two entries. The first one, `<authentication>`, sets the authentication mode for the solution. The second one, `<identity>`, indicates that impersonation is not currently enabled for this solution.

For testing purposes, you should put at least one test page in the same Web folder. Listing 20.1 is a simple DEFAULT.ASPX file you can use as an example.

Listing 20.1 DEFAULT.ASPX—Simple DEFAULT.ASPX to Test Windows Authentication

```
<%@ Page Description="windows-based security" %>

<script LANGUAGE="vb" RUNAT="server">

sub Page_Load(sender as Object, e As EventArgs)
  userID.Text=User.Identity.Name
  authType.Text=User.Identity.AuthenticationType
end sub

</script>

<html>
```

Listing 20.1 continued

<body>

<h2>Windows-Based Security Home</h2><hr />

User Identity: <asp:Label id="userID" runat="server" />

Auth Type: <asp:Label id="authType" runat="server" />

You passed Windows-Based security!

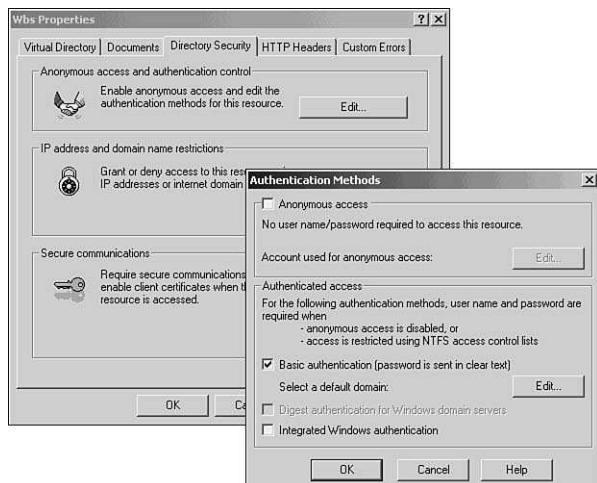
</body>

</html>

You should notice that the `Page_Load` event contains two calls to the `User.Identity` object. This will return the validated user name as well as the method IIS used to authenticate the user.

Now that all the proper ASP.NET content has been added, it's time to make adjustments to IIS to turn off anonymous access. This will force all users to successfully log into the machine when they request any page from the WBS site. All you need to do is call up IIS Manager, select the Web you wish to modify (WBS) and then right click to select the properties box for the Web, click on the "Directory Security" tab, and press the "Edit" button in the "Authentication Control" box.

Now you can check off the anonymous authentication and check on either Basic or Integrated Windows (NTLM) authentication (see Figure 20.4).

**Figure 20.4**

Setting IIS to support Windows authentication.

TIP

NTLM is the most secure authentication method, but only Microsoft Internet Explorer supports it. If you are supporting browsers other than Microsoft Internet Explorer, you should select both Basic and NTLM authentication modes. That way IIS will use NTLM for Microsoft Internet Explorer browsers and Basic for all others.

Now that you have adjusted IIS to force user authentication, you can use your browser to view any page in the secured Web. The first time you request a page at the Web server, you'll be asked to present your credentials in the form of a valid username and password pair. Figure 20.5 shows the results of successfully logging into an ASP.NET page secured with Basic authentication.

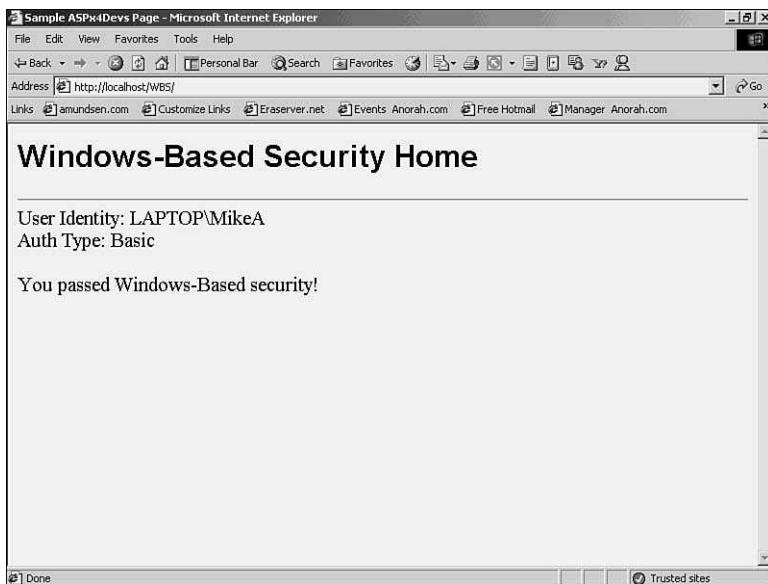


Figure 20.5

Results of successfully logging into WBS Web page.

NOTE

If you are running your test Web on your own local machine and you have selected NTLM, you'll not be asked for a login name and password. That is because the server already knows who you are.

Windows-Based Authorization

Now that you know how to enable Windows-based authentication, you can use Windows to create authorization entries to control access to server resources such as

files and folders. You accomplish this by making changes in the operating system ACLs, not by doing anything within the ASP.NET platform or within your code.

For example, you can create a new subfolder off the main directory (call it “SecuredFolder”) with a copy of the DEFAULT.ASPX Web page and set access security to include only authenticated users. Now, any user that passes the Windows authentication will be authorized to view the contents of the secured folder.

For another example, you can create a subfolder called “Private” and set the security to contain a single username from the Windows Users list. Now, when you successfully log into the site with another user name, any attempts to view the contents of the “Private” folder will result in an “access denied” message (see Figure 20.6).

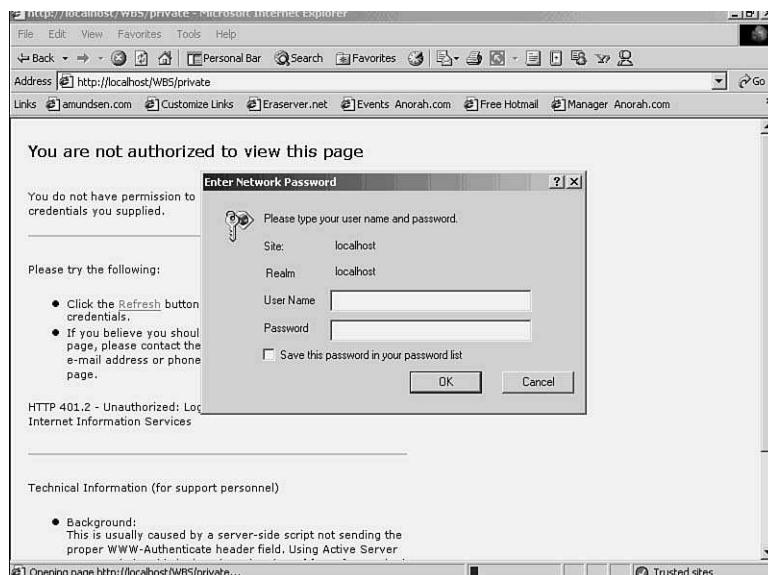


Figure 20.6

Attempting to access restricted content.

Implementing Forms-Based Security

While Windows-Based Security is a very powerful model, it is not always the most appropriate solution for Web applications. For example, WBS requires unique users to be defined within the operating system. This is not very practical for public Web sites that use login information to personalize content and layout on otherwise open Web sites. In this scenario, it is often better to use ASP.NET's Forms-Based Security (FBS). FBS allows you to define an ASP.NET input form to collect authentication data and then use your own custom data store to handle authorization processing at runtime.

The primary advantage of FBS is that you do not need to create unique users at the operating system level. This also makes it easy to create a portable site since moving

the content from one server to another does not require recreating long lists of users and ACLs on the target server in order to support the moved application. FBS also allows you to customize your authorization scheme to meet your application's needs without dealing directly with the operating system.

Of course, there are some downsides to FBS, too. First, FBS requires you to custom code both authentication capture and authorization processing. If done incorrectly, this can lead to an application that is not adequately secure or one that requires a great deal of coding and testing before final release. Also, as the solution grows in complexity, managing the authorization data can get to be a real chore.

Despite these minor drawbacks, FBS continues to be a viable and important way to secure your Web applications.

Forms-Based Authentication

The first step in building a forms-based security system is defining the security details in the configuration file for the Web. To do this, you can place `<authentication>` and `<authorization>` sections in the WEB.CONFIG file in the root folder of the Web application. For example, if you have a Web application called FBS, you'd place this content in the WEB.CONFIG file in the root folder. These sections will establish how users will be authenticated and who will be authorized to view content.

NOTE

When creating this application, you need to be sure to mark the top level folder (for example, FBS) as an IIS Web Application.

Listing 20.2 shows a complete WEB.CONFIG file that requires all users to successfully pass authentication before viewing any content in the Web.

Listing 20.2 WEB.CONFIG—Establishing Forms-Based Security for an ASP.NET Application

```
<configuration>

    <system.web>

        <authentication mode="Forms">
            <forms name="fbs" loginUrl="login.aspx">
                </forms>
        </authentication>

        <authorization>
            <deny users="?" />
        </authorization>

    </system.web>

</configuration>
```

Notice that the <forms> tag indicates the location of a login form. This is the form used to capture user credentials and test them against a user data store. For example, you can create a simple XML file to contain user names and passwords. Listing 20.3 shows an example of just such a file (USERLIST.XML).

Listing 20.3 USERLIST.XML—Example of User Credential Data Stored in an XML File

```
<users>
  <username>PaulL</username>
  <password>password</password>
  <username>MikeA</username>
  <password>password</password>
</users>
```

Now that you have defined both the authentication scheme (forms) and the authentication credentials list (USERLIST.XML), you need to create a login form to capture and validate the user input. Listing 20.4 shows the HTML markup for a typical user login form (LOGIN.ASPX).

NOTE

XML files are just one way to store user credentials. You can also store the data in an SQL database for some other structured storage system.

Listing 20.4 LOGIN.ASPX—HTML Markup for a Typical User Login Form

```
<h2>FBS Login Page</h2>
<hr />

<form runat="server">
  <table>
    <tr>
      <td>UserName</td>
      <td><asp:TextBox id="userName" runat="server" /></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><asp:TextBox id="password" textmode="password" runat="server" /></td>
    </tr>
    <tr>
      <td>Persist Cookie</td>
      <td><asp:CheckBox id="persistCookie" runat="server" /></td>
    </tr>
    <tr>
      <td>&nbsp;</td>
      <td><asp:Button id="submit" text="Login"
        Onclick="submit_click" runat="server" /></td>
    </tr>
```

Listing 20.4 continued

```
</table>
</form>

<font color="red">
    <asp:Label id="Status" runat="server" />
</font>
```

Once the markup is completed, you need to add server-side code to handle the submit_click event defined in the markup. There are three tasks to complete in order to properly authenticate users. First, you need to read the list of valid users. Next, you need to check the user inputs against that list. Finally, if the credentials are valid, you need to forward the user to the requested page. If not, you need to present a message to the user that the credentials are invalid. Listing 20.5 shows the complete server-side code needed to implement validation based on the HTML markup in Listing 20.4 and the USERLIST.XML file from Listing 20.3.

Listing 20.5 LOGIN.ASPX—Complete Server-Side Code to Implement Forms-Based User Authentication

```
<%@ Page Description="forms-based security" %>
<%@ Import Namespace="System.Web.Security" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.IO" %>

<script LANGUAGE="vb" RUNAT="server">

dim ds as DataSet = new DataSet()

sub submit_click(sender as object, e as EventArgs)

    readUserFile
    if validateUser(username.Text)=true then
        FormsAuthentication.RedirectFromLoginPage(userName.Text, _
            persistCookie.Checked)
    else
        Status.Text="Invalid Login!"
    end if

end sub

sub readUserFile

    dim fs as FileStream = new FileStream(Server.MapPath("userList.xml"), _
        FileMode.Open, FileAccess.Read)
    dim reader as StreamReader = new StreamReader(fs)
    ds.ReadXml(reader)
    fs.Close()

end sub
```

Listing 20.5 continued

```
end sub
function validateUser(byVal username as string) as Boolean

    dim search as string = "username_Text=''' & username & '''"
    dim userTable as DataTable = ds.Tables(0)
    dim matchList() as DataRow = userTable.Select(search)

    if matchList.Length>0 then
        return(true)
    else
        return(false)
    end if

end function

</script>
```

In Listing 20.5, the method `readUserFile` loads all the data from the `USERLIST.XML` file into a `DataSet` object. The `validateUser` method then searches that `DataSet` to find the `username` entered into the form. If the `username` is found, the method returns `TRUE`. In this example, the password is never checked in order to keep the code simple. In a production application, you'd need to check the password, too.

Finally, if the `validateUser` method returns `TRUE`, then the `FormsAuthentication.RedirectFromLoginPage` method is used to write an authentication cookie to the header and then forward the user to the requested page. If the method returns `FALSE`, then the “Invalid Login” message is displayed on the form.

Now, whenever anyone requests a resource from the Web, they will first be directed to the `LOGIN.ASPX` page. If they enter valid credentials, they will automatically be forwarded to the proper page. It is important to point out that no validation code is needed in the actual Web pages that users will view. Only the `LOGIN.ASPX` page needs the code. The `ASP.NET` runtime will handle the rest.

Forms-Based Authorization

You can use additional `WEB.CONFIG` files in subfolders of the solution to implement a more complex authorization scheme. For example, you can add a subfolder called `PRIVATE` to the root and a `WEB.CONFIG` file that denies access to one or more authenticated users. This will allow only selected users to view the contents of the subfolder.

The following code shows an example `WEB.CONFIG` that you would place in the `PRIVATE` folder to prevent the authenticated user, `MikeA`, from viewing the folder contents.

```
<configuration>
    <system.web>
```

```
<authorization>
    <deny users="MikeA" />
    <allow users="?" />
</authorization>

</system.web>

</configuration>
```

It is important to note that even though the user, MikeA, is an authenticated user in the Web, that user is still not allowed to view the contents of the PRIVATE folder.

Summary

In this chapter you learned a number of key concepts related to implementing successful security schemes for ASP.NET Web applications including

- Authentication—Determining the identity of the user requesting resources.
- Authorization—Checking to see if the authenticated user has the proper rights to the requested resource.
- Impersonation—Establishing the Windows identity used to execute your application code.
- Windows Based Security (WBS)—Using the existing Windows access control list (ACL) to establish the authentication of users.
- Forms Based Security (FBS)—Using your own list of users and passwords to authenticate users.

You also learned how IIS and ASP.NET work together to complete user authentication and authorization.

CHAPTER 21

Configuring and Deploying Your ASP.NET Solutions

Now that you know how to build ASP.NET Web Forms, create data-bound applications, support Web services, and design secure solutions using ASP.NET security models, you're ready to learn how to configure and deploy your Web solutions to your production servers.

In the past, configuring and deploying Web solutions was tedious at best, and complicated at worst. This was especially true when your solutions contained one or more compiled components. These components often relied on detailed registry settings, were cumbersome to install remotely and were even more difficult to update once the original component was placed into production.

ASP.NET solutions are much easier to configure since they no longer require registry settings, but instead use a standardized XML-based configuration file to hold all important application data. In addition, ASP.NET greatly simplifies the deployment and maintenance of Web solutions since compiled components no longer require registry support and can easily be updated simply by copying new components over old ones—even while the old ones are still running on the production server.

In this chapter, you learn how to configure your Web solutions using ASP.NET's standardized configuration files. You also learn how to design deployment plans that make it easy for you to install your ASP.NET solutions both locally and remotely.

Configuring Your ASP.NET Solutions

ASP.NET solutions are designed to use a standardized XML-based configuration file for all important runtime settings. In fact, the ASP.NET runtime uses this same kind of configuration file to control the runtime behavior of ASP.NET on the server.

This standardized XML file can be viewed and updated with any simple editor such as Microsoft Notepad or other tool. And, although the configuration file is a standard format, since it's in XML form, the configuration file is easily extended to support custom configuration information needed to support your own application.

The following sections in this chapter show you how the configuration file is arranged, how the ASP.NET runtime uses the configuration data, and how you can use configuration files to control how your ASP.NET solutions behave at runtime.

Understanding the ASP.NET Configuration Model

The ASP.NET configuration model is based on an XML configuration file. Every server that has the ASP.NET platform installed on it contains at least one of these files, called MACHINE.CONFIG. In the default installation of the .NET platform, the MACHINE.CONFIG is kept in the following folder:

C:\WINNT\Microsoft.NET\Framework\v1.0.2615\CONFIG

The MACHINE.CONFIG File

The MACHINE.CONFIG file contains all the settings that control how the entire ASP.NET runtime behaves. Also, this MACHINE.CONFIG contains the default settings for all other Web applications installed on the server. The basic layout of the MACHINE.CONFIG is shown in Listing 21.1.

Listing 21.1 SAMPLE.CONFIG—A Sample ASP.NET Configuration File

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

<configuration>

    <system.web>
        <trace
            enabled="false"
            requestLimit="10"
            pageOutput="false"
            traceMode="SortByTime"
            localOnly="true"
        />

        <pages buffer="true"
            enableSessionState="true"
            enableViewState="true"
            enableViewStateMac="true"
            autoEventWireup="true" />
```

Listing 21.1 continued

```

<customErrors mode="RemoteOnly" />

<sessionState
    mode="inproc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user id=sa;password="
    cookieless="false"
    timeout="20"
/>

<iisFilter
    enableCookielessSessions = "true"
/>

</system.web>

</configuration>

```

Actually, there are many more sections in the master MACHINE.CONFIG for an ASP.NET server. The example in Listing 21.1 shows just a few of the important sections. You'll learn more about the possible configuration sections later in this chapter. The important thing to note is that the file is in familiar hierarchical XML format and supports a number of sections.

NOTE

The CONFIG files stored on the Web server are protected by security policies within ASP.NET. For example, any time a user requests a CONFIG file via the browser, the ASP.NET service will return a 403 error (forbidden request).

Configuration Inheritance

A major feature of the ASP.NET configuration system is that it supports a form of inheritance. In other words, settings in the MACHINE.CONFIG are inherited by all running Web applications on that machine. The only time this is not the case is when the individual Web applications have their own CONFIG files that override the MACHINE.CONFIG. For example, let's say the MACHINE.CONFIG file has the following setting:

```
<pages enableSessionState="true" enableViewState="true" />
```

This means that for all Web applications on this machine, session state is enabled and view state is enabled. Next, let's say a Web application on the same machine has its own configuration file, called WEB.CONFIG, with the following setting:

```
<pages enableSessionState="false" />
```

First, notice that the enableViewState attribute does not appear in the Web's configuration file. That means that the local Web will "inherit" the setting from the MACHINE.CONFIG file for the server. Thus, for this particular Web, session state is disabled, and view state is enabled.

Finally, configuration information can also be kept in WEB.CONFIG files in sub folders within a single Web application. This means that you can customize the application behavior by sub folder, too. For example, if the root folder of the Web application has an authorization section that allows all users to view the contents of the root Web, then you can do this:

```
<authorization>
    <allow users="*" />
</authorization>
```

However, in a subfolder off the root Web, called "SECURE", there is another configuration file that has a setting to allow only validated users to view the contents of the sub folder:

```
<authorization>
    <allow users="?" />
</authorization>
```

This means that any attempts by users to view the contents of the SECURE folder will result in a request for user credentials. If the system determines that the credentials are valid, the user will be allowed to view the contents of the folder.

Configuration Elements

Since the ASP.NET CONFIG files are extensible, the actual contents of the files can vary greatly. However, there are several basic elements that could often be found in ASP.NET CONFIG files. For example, the MACHINE.CONFIG file contains the following top-level sections:

- **ConfigSections:** This section contains definitions for all the different configuration sections found in the file along with .NET components that are to be used to interpret the contents of each section.
- **AppSettings:** This is an open section that can be used by ASP.NET programmers to place name/value pairs for access at runtime. This is an ASP.NET version of registry settings for Web applications.
- **system.diagnostics:** This section contains information used by the ASP.NET runtime to perform internal diagnostic routines.
- **system.net:** This section contains information on network services for ASP.NET.
- **system.web:** This section contains information used to set the behavior of all Webs on the machine.
- **runtime:** This section contains information about the actual .NET runtime system.

Theoretically, virtually all of the sections in the MACHINE.CONFIG file could be overridden in WEB.CONFIG files within Web applications running on the same machine. However, many of the MACHINE.CONFIG files are of little interest to application-level programming. The primary exception to this rule is the <system.web> section. This is the section that defines the behavior of running Web applications and it is often overridden in the local WEB.CONFIG files.

Common Configuration Elements in WEB.CONFIG Files

The WEB.CONFIG file can be thought of as ASP.NET's version of the registry settings for traditional Windows applications. You can use the WEB.CONFIG file to store important settings that can be accessed at runtime. Following is a list of commonly used sections in WEB.CONFIG files including notes on the contents of these sections.

AppSettings

This is a general top-level section that can be used to store name value pairs for use in your application. For example, you can store database connection information here, common folders or file pointers, even extended information such as default values for dialogs, etc. (See Listing 21.2)

Listing 21.2 COMMON_SECTIONS.CONFIG—Example AppSettings Section

```
<configuration>
<appSettings>
  <add key="PubsDSN" value="server=myServer;database=pubs;userid=sa;
  =>password=''" />
  <add key="dialogLevel" value="expertMode" />
  <add key="defaultBackgroundColor" value="white" />
  <add key="defaultFontColor" value="black" />
</appSettings>
</configuration>
```

sessionState

This section, occurring within the <system.web> section, controls the session state behavior of the application. Following is a typical session state entry:

```
<sessionState
  mode="inproc"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConnectionString="data source=127.0.0.1;user id=sa;password="
  cookieless="false"
  timeout="20"
/>
```

The mode setting of sessionState can be set to *inproc*, *stateserver*, or *sqlserver*. If mode is set to “*inproc*,” then all data is kept on the local machine. When mode is set to “*stateserver*,” all session data is kept on a single shared server in the network. When mode is set to “*sqlserver*,” all session data is stored within a SQL Server database. If the cookieless attribute is set to true, session data can be tracked even if the client browser has cookie support turned off.

customErrors

The customErrors setting, part of the <system.web> section, allows you to control how and when a customized error page is displayed to users. The following listing shows a version of the custom errors section of WEB.CONFIG.

```
<system.web>
  <customErrors mode="On" defaultRedirect="errorGeneric.aspx">
    <error statusCode="404" redirect="notfound.aspx"/>
    <error statusCode="403" redirect="notallowed.aspx"/>
    <error statusCode="500" redirect="apperror.aspx"/>
  </customErrors>
</system.web>
```

The default setting is “remoteOnly.” This means custom errors will only be displayed when users are calling from a remote machine. This allows local calls to see the standard Internet Information Server error information, while remote users will see a more friendly error page.

There are a number of possible configuration sections for WEB.CONFIG files. Check the ASP.NET documentation for a comprehensive list of configuration sections and settings.

Accessing Configuration Data at Runtime

You can access values in the WEB.CONFIG file of your Web application at runtime. In this way, you can store important values such as data connections and other configuration information in the file and then read them into your program as needed.

Listing 21.3 shows the code that can be used to read in all the values in the appSettings section of the WEB.CONFIG file.

Listing 21.3 DEFAULT.ASPX—Reading the Contents of the AppSettings Section of a WEB.CONFIG File

```
<%@ Page Description="Accessing Configuration Settings" %>

<script language="vb" runat="server">

sub Page_Load(sender as object, args as EventArgs)

  dim strKey as string

  for each strKey in ConfigurationSettings.AppSettings
    configDisplay.Text &= "<b>" & strKey & "</b> :: "
    configDisplay.Text &= ConfigurationSettings.AppSettings(strKey)
    configDisplay.Text &= "<br />"
  next

end sub
</script>
```

Listing 21.3 continued

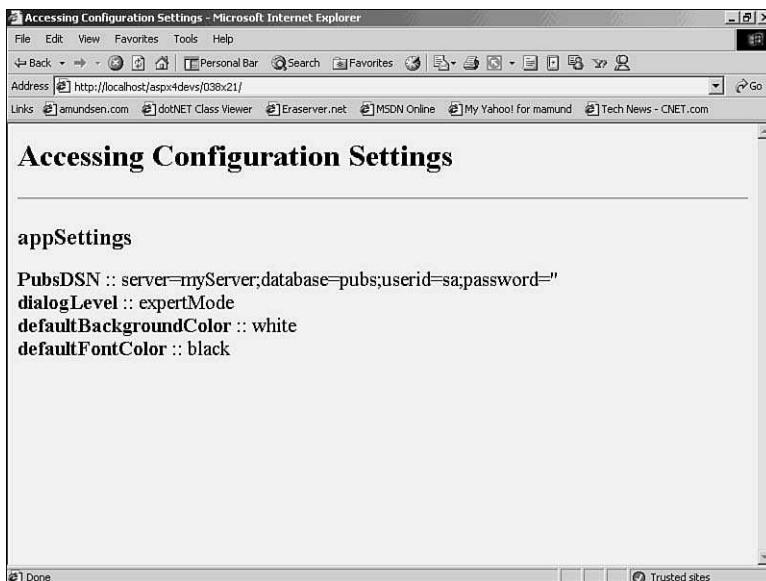
```
<html>
<body>

<h2>Accessing Configuration Settings</h2>
<hr />

<h3>appSettings</h3>
<asp:Label id="configDisplay" runat="server"/>

</body>
</html>
```

When you run this code against the WEB.CONFIG file that is shown in List 21.2, you'll see the list of appSettings values appear in the browser (See Figure 21.1).

**Figure 21.1**

Viewing the appSettings values from WEB.CONFIG in a browser.

Deploying Your ASP.NET Applications

Deploying ASP.NET solutions is actually quite easy, even if you are using compiled components in your solution. In the past, in order to completely install Web solutions, you needed to have access to the Internet Information Server Manager, COM+ Services, and even have the rights to update the registry. This was made even more difficult if you needed to deploy your solution on one or more remote servers.

Now, with ASP.NET, you can simply create the target folder(s) and copy the files directly to the target. If you have compiled components in your solution, you only need to be sure to place them in the proper subfolder. The rest is handled by ASP.NET. There is no need to attempt to register the components using the old REGSVR32.EXE.

Even better, if you need to update these compiled components on a production server, you no longer need to stop Internet Information Server or bring down the actual machine. All you need to do is copy the new component(s) over the existing one(s) and all will be taken care of for you.

In this section, you learn how to successfully move your Web pages and components to the target server and ASP.NET is able to automatically register and update your solution components. You also learn how to create scripts that handle most all of these tasks remotely.

TIP

If you are using Microsoft Visual Studio.NET to build and deploy your solutions, you may not need to go into the details of creating virtual folders, and copying files to your target server. Instead, you can use one of the many built-in deployment features of Visual Studio.NET.

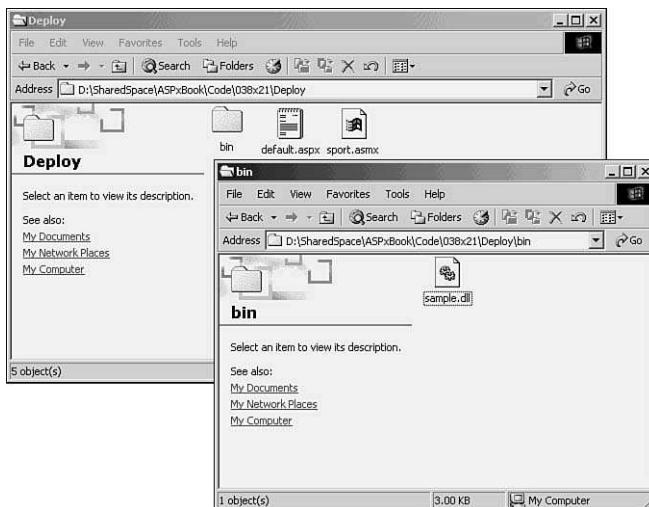
The New Deployment Model for ASP.NET Solutions

The ASP.NET deployment model is based on simple file copy technology. This means that you don't need any special utilities or packaging tools in order to deploy ASP.NET Webs to the target server. This "minimal deployment" model is made possible by some special services running with the .NET platform. These services are smart enough to handle the process of locating and managing compiled components on the ASP.NET server. These services are so effective that you no longer need to register your components in the Windows registry at all.

The heart of this deployment model is that all DLLs for your Web need to be placed in a special folder, the BIN folder, which is a subfolder off the root folder of your solution. For example, imagine you have a root folder called DEPLOY that contains the documents DEFAULT.ASPX and SPORT.ASMX. You also have a single compiled component, called SAMPLE.DLL that is called by both the DEFAULT.ASPX and SPORT.ASMX documents (see Figure 21.2)

Notice in Figure 21.2 that the SAMPLE.DLL file is in a subfolder called BIN. Since ASP.NET does not use the Windows Registry, the only way that ASP.NET can locate your custom DLLs is for you to place them in the BIN subfolder of your solution. If you fail to place them in the correct place, you'll receive a runtime error telling you that ASP.NET cannot locate your component.

This is the only requirement that ASP.NET puts on the physical layout of your solution. You can add as many folders and files as you wish to the solution as long as all compiled DLLs are always placed in the same folder.

**Figure 21.2**

Sample Web files and folders.

Now that you know how easy it is to deploy compiled components, you're ready to learn the details of how you can create the folders and copy the files to both local and remote ASP.NET Web servers.

Copying the Application to the Web Server

The first step in deployment is to create the proper folders on your target ASP.NET machine and copy all the proper files from the source machine to the target machine. This can be done using a simple XCOPY command or, for remote server, you can use FTP commands. If you have a GUI file explorer, you can even use simple drag and drop steps to complete the work. The important thing to keep in mind is that you only need to copy the files to the target machine in order to properly build and register the ASP.NET solution.

The files you need to copy depend on the types of components you have in your solution. These include your ASPX (and related) files along with all images and other resource files.

Following is a list of commonly used ASP.NET files that you should deploy to your target server if they exist in your source project:

- .ASPX: Web forms files
- .ASMX: Web services files
- .ASCX: user control files
- .ASHX: ASP.NET handler files
- .CONFIG: configuration files
- .VB or .CS: only if they are used as code-behind files for related .ASPX pages
- DLL: .NET compiled components (in the BIN folder)

Using an XCOPY Script

Probably the simplest way to deploy our ASP.NET solution is to use the command line program XCOPY to copy the files from your source machine to the target machine. This is a viable solution if you have a mapped drive or UNC path to the target server from the source machine.

For example, imagine you have a mapped drive Z: that represents the root of the Web server and you want to copy files from the DEPLOY folder on your machine to the TARGET folder on the Web server. Here is an example XCOPY script that can handle the task:

```
xcopy c:\deploy\*.* z:\target /V/Q/Y  
xcopy c:\deploy\bin\*.* z:\target\bin /V/Q/Y  
pause
```

This script first copies all the files from the DEPLOY folder of the local machine to the TARGET folder on the remote machine. Next, all the files from the DEPLOY\BIN folder are copied to the TARGET\BIN folder. Although you could actually use a single XCOPY command (xcopy c:\deploy*.* z:\target /s) to copy all files from the source to the destination, the above command skips folders that might contain source code for components, design documents and other non-Web files. The PAUSE statement is included only to allow the user to view the results of the command line execution before dismissing the command prompt dialog box.



TIP

If you can reach the target machine via mapped drives, you can probably just use the Windows File Explorer to simply drag the proper files and folders from the source machine to the target machine.

Using FTP Services

If the target machine is not reachable via a mapped drive, you can probably use FTP services to copy the files and folders to the target location. This requires that you have a valid FTP account at the target machine, that your account has rights to write new files on the target machine, and that the target machine publish an FTP entry point for you such that you can reach the target folder.

The following listing shows a sample set of FTP commands that can move the contents of the DEPLOY and DEPLOY\BIN folders from the local machine to the remote machine.

```
open localhost  
user aspx4devs smart  
prompt off  
cd target  
mkdir bin  
mput d:\sharedspace\aspxbook\code\038x21\deploy\*.*  
cd bin  
mput d:\sharedspace\aspxbook\code\038x21\deploy\bin\*.*  
bye
```

This example uses the FTP.EXE utility that ships with the Windows operating system. There are also a number of third-party FTP tools that offer rich user interfaces and scripting tools to make this process easier.

**TIP**

If you have a valid FTP account that supports uploads for your Webs, you can probably use the Microsoft Internet Explorer to FTP your files to the target. Just load Microsoft Internet Explorer and enter the FTP site you wish to contact. You can then drag and drop your files into place.

Creating the Virtual Directories on the Web Server

As mentioned earlier, all ASP.NET solutions require an entry point on the Web server. This entry point is called the “virtual directory.” It is the starting point for your solution. Typically, the DEFAULT.ASPX page and many other documents are placed in this folder.

Also, if your solution has one or more precompiled components, you need to place them in a folder called BIN that is located just below the root folder of the virtual directory. This will make sure that ASP.NET can locate the compiled components at runtime.

The first step is to create the folders on the target machine. Once this is done, you need to mark the starting folder of your solution as virtual directory so that Internet Information Server will be able to route user requests to your ASP.NET application.

Creating the folders is easy. You can use simple DOS commands directly or through a script. Often you can just use your Microsoft File Explorer tool to browse to the target server and add folders as needed. You can also use FTP utilities on remote servers to do the same thing.

The important point to remember is that, once you create the folders and copy the files, you need to mark the base level as a “virtual directory.” You can do this a number of different ways. The three ways covered in this chapter are

- Using File Explorer
- Using Internet Information Server
- Using a WSH Script

Using File Explorer

The simplest way to create a virtual directory on your local machine is to browse to the selected folder using the Microsoft File Explorer and complete the following steps:

1. Locate the desired folder on the machine using Microsoft File Explorer.
2. Right click over the desired folder to view the context menu.
3. Select ‘Properties’ from the context menu.
4. When the dialog appears, select the ‘Web Sharing’ tab (see Figure 21.3).

5. Click on the ‘Share this folder’ radio button.
6. When the ‘Edit Alias’ dialog appears, verify that the proper name appears and press the OK button until you dismiss all dialogs.

Figure 21.3 shows how steps 4 through 6 would look on a Windows 2000 machine.

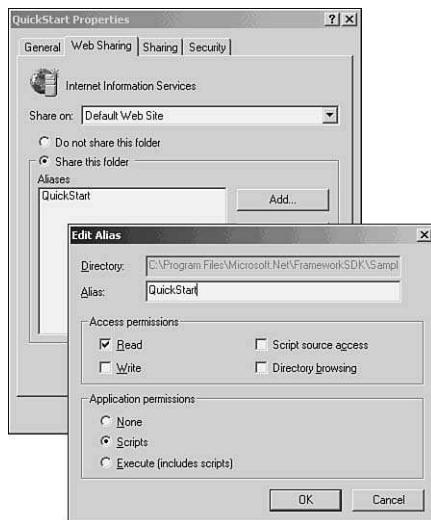


Figure 21.3

Using File Explorer to create a Web share on the server.

While this method is very easy, it has its limits. First, you can only do this if you are logged onto the machine locally. For example, you cannot perform this task on a shared network disk. Second, the dialogs only let you do the simplest kind of Web publishing. If you need to adjust Internet Information Server rights, apply certificates, and so on, you need to use Internet Information Server Manager instead.

Using Internet Information Server Manager

Probably the most comprehensive way to create a new Web share is to use the Internet Information Server Manager utility. This very powerful utility walks you through the process of creating and managing multiple Webs on multiple servers. Also, if configured properly, the Internet Information Server Manager can be used to manage any number of remote servers.

The details of creating and managing Webs with Internet Information Server Manager are beyond the scope of this book. However, all you need to do is to load the Internet Information Server Manager, expand the desired server in the left-side tree and right-click over the machine, and select “New—Virtual Directory” from the context menu. This will call up the “Virtual Directory Creation Wizard” that will walk you through the process from start to finish (see Figure 21.4).

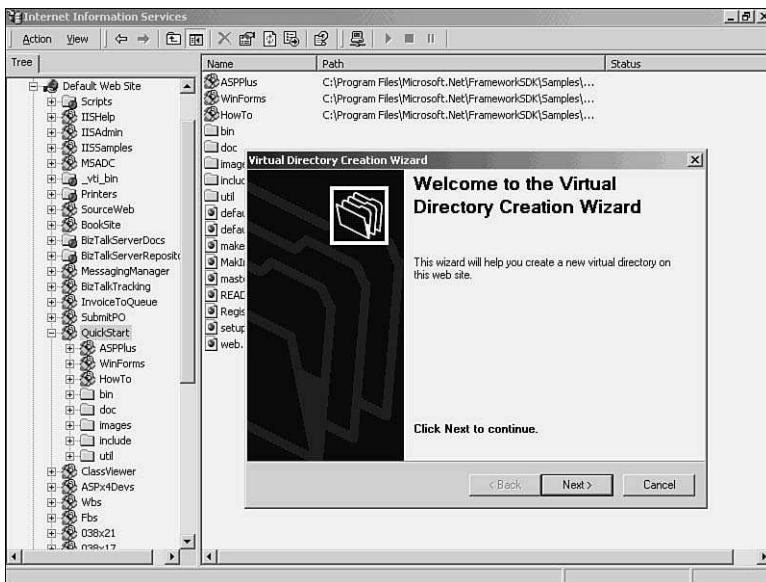


Figure 21.4

Using Internet Information Server Manager to create a Web share.

The WSH Script

While both the File Explorer and Internet Information Server Manager solutions are valuable, sometimes you need to create a new virtual directory on several servers at a time or you wish to create the directories without having to click through all the dialogs. For that you need a scripted solution. Luckily, Internet Information Server ships with a number of management scripts to handle just such tasks.

Listing 21.4 shows the start of an example WSH script that can create the desired Web on the target machine without using any UI at all. This is especially valuable if you need to deploy your solution to a series of servers in a Web farm.

Listing 21.4 MAKETARGETWEB.VBS—Sample WSH Script to Create a Web Share

```
Option Explicit
dim vPath, scriptPath,vName

vName="Target" ' name of web to create

' get current path to folder and add web name to it!
scriptPath = left(Wscript.ScriptFullName, _
len(Wscript.ScriptFullName) -len(Wscript.ScriptName))
vPath = scriptPath & vName

'call to create vDir
CreateVDir(vPath)
```

Listing 21.4 continued

```
' code taken from mkwebdir.vbs and changed for single vDir creation.  
Sub CreateVDir(vPath)  
  
    Dim vRoot,vDir,webSite  
    On Error Resume Next  
  
        ' get the local host default web  
        set webSite = findWeb("localhost", "Default Web Site")  
        if IsObject(webSite)=False then  
            Display "Unable to locate the Default Web Site"  
            exit sub  
        else  
            'display webSite.name  
        end if  
  
        ' get the root  
        set vRoot = webSite.GetObject("IISWebVirtualDir", "Root")  
        If (Err <> 0) Then  
            Display "Unable to access root for " & webSite.ADsPath  
            Exit sub  
        else  
            'display vRoot.name  
        End IF  
  
        ' delete existing web if needed  
        vRoot.Delete "IISWebVirtualDir",vName  
        vRoot.SetInfo  
        Err=0 ' reset error  
  
        ' create the new web  
        Set vDir = vRoot.Create("IISWebVirtualDir",vName)  
        If (Err <> 0) Then  
            Display "Unable to create " & vRoot.ADsPath & "/" & vName & "."  
            exit sub  
        else  
            'display vdir.name  
        end if  
  
        ' set properties on the new web  
        vDir.AccessRead = true  
        vDir.Path = vPath  
        vDir.AccessFlags = 529  
        VDir.AppCreate False  
        If (Err <> 0) Then  
            Display "Unable to bind path " & vPath & " to " & _  
vRoot.Name & "/" & vName & ". Path may be invalid."  
            exit sub  
        end If
```

Listing 21.4 continued

```
' commit changes
vDir.SetInfo
If (Err <> 0) Then
    Display "Unable to save changes for " & vRoot.Name & "/" & vName & "."
    exit sub
end if

' report all ok
WScript.Echo Now & " " & vName & " virtual directory " & vRoot.Name & "/"
-> & vname & " created successfully."
End Sub
```

The script in Listing 21.4 is just the top-most routines of a much longer script. It is not important to understand all the details of this script. Instead, it's important to know that you can create a fully scripted solution for creating virtual directories on target servers.

Handling Component Registration, Versioning, and Replacement

Once you have your ASP.NET Web deployed and running in production, you're bound to discover that you need to update one or more documents or components on the production server. In the past, this meant that you needed to stop the Internet Information Server service on the machine. Sometimes you even needed to reboot the server itself before you could complete your updates.

However, ASP.NET allows you to update both pages and compiled components without stopping any services on the machine at all. If you have new documents or DLLs to place on the production machine, all you need to do is perform an XCOPY or FTP PUT command to place the new files on the server. Once ASP.NET detects that you have placed new files on the server, the runtime services will flush the old components from memory and load the new components instead. From that point on, all site users will see the new pages and components.

ASP.NET is even smart enough to maintain a copy of the old components as long as there is at least one site visitor running the old components. In other words, when you copy new files over old, ASP.NET will not simply drop any existing users and cancel their sessions. Instead, ASP.NET will maintain existing user's paths through the old components until their current HTTP request is completed. At the same time, ASP.NET will route all new HTTP requests to the updated components. Once all existing HTTP requests are drained from the old components, these old files will be removed from memory and deleted from the machine.

This means you can safely update both pages and compiled DLLs on your production machine with minimum disruptions to your users.

Summary

ASP.NET uses the XML-base CONFIG files to control both the ASP.NET runtime and the runtime behavior of your Web applications. You now know how to modify the CONFIG files and how to access the contents of the WEB.CONFIG file at runtime.

You also learned how to deploy your completed ASP.NET solution using XCOPY or FTP and how to create virtual directories using either operating system utilities or simply by creating a WSH script. ASP.NET will allow you to easily update the pages and DLLs in your solution without having to restart Internet Information Server or reboot the server.

APPENDIX A

Moving to VB .NET from VB6 or VBScript

Visual Basic .NET (VB .NET) is an exciting new version of Visual Basic. Unfortunately, if you have any experience with a prior version of VB or VBScript, it may take some time to adjust to this radically different version of VB. This appendix highlights the significant changes in VB .NET and offers some advice to ease the move.

What's New in VB .NET?

So much has changed that we could easily spend 50, 100, or more pages discussing the changes to VB .NET. Rather than be comprehensive, our goal in this appendix is to merely highlight the major changes in VB .NET from the viewpoint of the developer with prior VB6 or VBScript programming experience.

NOTE

See "A Programmer's Introduction to Visual Basic .NET" (ISBN 0672322641) by Craig Utley for a more complete discussion of moving to Visual Basic .NET. A briefer electronic version of this text is available for free at [www.samspublishing.com/detail_sams.cfm?item=067232203X](http://samspublishing.com/detail_sams.cfm?item=067232203X)

Moving from VBScript/ASP to VB .NET

If you're primarily an ASP VBScript developer, your upgrade to VB .NET should be fairly smooth. In most cases, it's just a matter of learning what new features are now at your disposal that weren't available in the VBScript language.

Much of this appendix applies to developers upgrading from both VB6 and VBScript. Here's a list, however, of the issues that are most relevant to the former VBScripter (most issues are discussed in more detail in later sections of this appendix):

- **Option Explicit.** Many ASP developers never used `Option Explicit`, in part because there was no documentation on how to employ it in your ASP pages. You should now use it or the more rigorous `Option Strict` on all pages.
- **Data types.** VBScript lacked support for variable data types. You should start using VB .NET data types as soon as possible.
- **Employing the .NET Framework.** A number of VBScript features have been removed from the language and replaced with properties and methods of the .NET Framework. See the later section "Where Is...?" for more details.
- **Object-Oriented Programming (OOP) Features.** Although Microsoft added the `Class` statement to VBScript 5.0, most ASP developers did not use it. VB .NET supports classes and OOP in a big way. See the later section, "VB does OOP" as well as Chapter 4, "Understanding Visual Basic .NET Syntax and Structure," for more details.
- **Financial functions.** VBScript lacked any financial functions. VB .NET now has full support for functions like `PMT`, `NPer`, and `IRR`.
- **Try/Catch error handling.** VBScript had very poor error handling support by virtue of `On Error Resume Next`. VB .NET now supports the much better Try/Catch error handling. See the later section "Try/Catch Error Handling" for more details.
- **Set no more.** The `Set` keyword is no longer required or supported for setting object variable references. See the later section, "Tightening the Reins," for more details.
- **Arrays.** The way you declare arrays has changed a bit, as have a few other array features. See "Arrays" later in this appendix for more details.

Moving from VB6 to VB .NET

VB6 developers may have more trouble upgrading to VB .NET than VBScript developers. That's because so much has changed. In this section, we point out the issues that are most relevant to the VB6 developer. All of these issues are discussed in later sections of this appendix:

- **Data types.** Data types in VB .NET have changed. Also changed is the way parameters are passed (they're now passed by value, by default). See the later section, "Tightening the Reigns," for more details.
- **Employing the .NET Framework.** A number of VB6 features have been removed from the language and replaced with properties and methods of the .NET Framework. See the later section "Where Is...?" for more details.
- **OOP Features.** While VB6 supported class modules, it lacked support for inheritance and other more advanced OOP features. VB .NET includes full support for OOP features. See the later section, "VB does OOP," as well as Chapter 4.

- **Try/Catch error handling.** VB .NET now supports Try/Catch error handling, which is almost always a better way to do things than using `On Error GoTo` (although this older error handling is still supported in VB .NET). See the later section, “Try/Catch Error Handling,” for more details.
- **Default Properties and Methods.** VB .NET no longer supports default properties and methods! In addition, the `Set` keyword is no longer required or supported for setting object variable references. See the later section “Tightening the Reins” for more details.
- **Arrays.** A lot has changed with arrays. You can no longer specify the lower bound of an array. In addition, there are other subtle changes. See “Arrays” later in this appendix for more details.

Tightening the Reins

One major goal of Microsoft in creating VB .NET was to clean up the aging Visual Basic language. A descendent of Microsoft’s GWBASIC for MS-DOS from the early 1980s, VB has suffered for some time because of its legacy.

In the interest of cleaning up and modernizing the language, Microsoft has made a number of significant changes to the VB language, including

- **Integer grows up.** For years, developers have struggled with the fact that SQL Server, ADO, and other databases used 32-bit integers, whereas VB used 16-bit integers. Microsoft has finally bitten the bullet and made VB .NET’s `Integer` into a 32-bit integer. Thus, VB .NET’s `Integer` data type can now hold what VB6’s `Long` held. At the same time, VB .NET’s `Short` is now equivalent to the old VB `Integer`. Finally, VB .NET’s `Long` can now hold 64-bit integer values.
- **Variant no more.** The chameleon of data types, `Variant`, has been ditched. The `Object` data type, however, now takes the place of `Variant`. If you need to create a general-purpose variable that can store different types of data, use `Object`. (But only use `Object` when you need to.)
- **Bye, bye default properties and methods.** One of the changes that is likely to trip up existing VB programmers is the elimination of default properties and methods. In VB6, for example, to set the value of a textbox you could use the following syntax because `Text` was the default property of a textbox:

```
textbox = "hello world"
```

In VB .NET, however, the above is illegal. Instead, you need to specify the `Text` property, as in the following:

```
textbox.Text = "hello world"
```

- **Set no more.** Due to the elimination of default properties and methods, VB .NET no longer requires the use of the `Set` keyword to set a reference to an object. For example, in VB6 and VBScript, you would use the following syntax to create an ADO Connection object:

```
Set cmd = New ADODB.Command
```

In an ASP page using VBScript, you might use the following syntax instead:

```
Set cmd = Server.CreateObject("ADODB.Command")
```

In both cases, you had to use `Set`. In VB .NET, you can (in fact, you *must*) drop the `Set` keyword entirely. So in VB .NET, the VB6 statement becomes (if you were using the older ADO library):

```
cmd = New ADODB.Command
```

This is a welcome change to VB that could only occur because of the elimination of default properties and methods.

- **ByVal over ByRef.** Prior versions of VB passed parameters to and from procedures by reference (`ByRef`); that is, by passing a pointer that points to the variable to the procedure. VB .NET, by default, passes parameters by value (`ByVal`); that is, by passing a copy of the variable to the procedure. You can still create `ByRef` parameters by including the `ByRef` keyword when you define the parameter.
- **Losing antiquated features.** A number of older, seldom-used statements and keywords have been retired. See the later section “Where Is...” for more details.

A stricter VB language will pay off in reduced programming bugs, but may cause some pain if you have to maintain older applications.

VB Does OOP

It was promised to be added soon in the early days of VB 1.0, but now complete support for inheritance and other OOP features has finally been added to the VB language. We've summarized some of these new features here:

- **VB “inherits” the world.** This is the big one that so many programmers have been asking Microsoft to add to the language for years. VB programmers can now create classes that inherit from other classes. But it's even better than that. You can now create VB classes that subclass (inherit from) C++ and C# classes (and vice versa). Subclasses (or derived classes) can inherit and extend the properties and methods of a parent class. Furthermore, a subclass can override the methods of a parent class.
- **Overloading methods.** VB .NET supports the overloading of properties and methods. This means that you can now create properties and methods of class modules with different data types and VB .NET will call the appropriate property or method based on the data type of the passed arguments.
- **Interface support.** VB .NET now supports the definition of interfaces. The use of interfaces allows you to separate the interface of a class from its implementation.

VB .NET's support for inheritance, overloading, and interfaces goes a long way to making a VB a true object-oriented language on par with OOP languages such as C++.

Arrays

Arrays have undergone some major changes in VB .NET. These changes were necessary to make arrays interoperable across the .NET languages. The changes include

- **All arrays have a 0 lower bound.** In prior versions of VB, you could define the lower bound of an array. For example, the following VB6 code declares an array named Company with 11 elements, numbered from 10 through 20:


```
Dim Company(10 To 20) As String
```

 This is no longer allowed.
- **No more Option Base.** Prior versions of VB allowed you to alter the default lower bound of arrays to 1 using the `Option Base` statement. The `Option Base` statement is no longer supported in VB .NET.
- **No more fixed-size arrays.** In prior versions of VB you could create either a fixed size array or a dynamic array. In VB .NET all arrays are dynamic. When you first declare an array, you can specify the initial number of elements using the `Dim` statement, but you can always change the size of a dimension later using the `ReDim` statement. VB .NET also differs from prior versions of the language in that you can no longer use `ReDim` to change the number of dimensions of an array; you may change only the number of elements within a dimension.

NOTE

See Chapter 4 for more details on VB .NET array functionality.

Try/Catch Error Handling

Error handling in VB has always felt like it was added to the language as an afterthought—a poorly conceived afterthought at that. VB6 and prior versions supported error handling via the `On Error GoTo` statement. (VBScript’s support was even worse.) While VB .NET still supports `On Error GoTo`, it also supports structured exception handling by borrowing the `Try...Catch...Finally` statement from C++.

For example, the following function attempts to open an ADO.NET SQL connection to the NorthWind SQL Server database, using a `Try...Catch...Finally` statement to handle any error encountered in opening the connection.

```
Function TestConnection() As Boolean
    Dim strCnx As String = _
        "server=localhost;uid=sa;pwd=;database=northwind"
    Dim cnxSQL As SqlConnection = New SqlConnection(strCnx)
    Dim e as Exception
    Dim blnReturn = True

    Try
        ' Attempt to Open the connection
        cnxSQL.Open()
    Catch e

```

```
' Write out error message
Response.Write(e.ToString)
blnReturn = False
Finally
    ' Close the Connection
    cnxSQL = Nothing
End Try

Return blnReturn
End Function
```

Other Changes

VB .NET includes a number of other nifty changes that you'll likely appreciate.

Dim Is Brighter

You can now declare and set the initial value of a variable in one fell swoop. For example:

```
Dim FirstName As String = "Anna"
```

Another change to Dim concerns the declaration of multiple variables in a single Dim (or Public, Private, Protected, or Friend) statement. For example, in VB6 the following code would create x and y as variants, and z as an integer:

```
Dim x, y, z as Integer
```

In VB .NET, the preceding creates all three variables—x, y, and z—as integers. In other words, it now works as it should!

New Assignment Operators

VB .NET includes several new assignment operators that simplify the setting of a variable to some transformation of itself. For example, if x = 5 and y = 2,

```
x += y
```

would add y to x, resulting in x = 7.

Similarly:

```
x -= y  ' after this statement, x = 3
x *= y  ' after this statement, x = 10
x /= y  ' after this statement, x = 2.5
x \= y  ' after this statement, x = 2
x ^= y  ' after this statement, x = 25
```

In addition, you can use &= (and +=) to concatenate strings. For example, if x = “Hello” and y = “World”:

```
x &= y
```

would result in x = “HelloWorld.”

Where Is...?

If you've been coding using the VB language for some time, you may be wondering where some of the features you have grown to love have gone. As mentioned in an earlier section, some older features have been killed, but many other features have just shifted from being part of the VB language to part of .NET Framework system classes.

In many cases, the older feature is still available but Microsoft encourages you to move to the new way of doing things. In other cases, the older way is gone completely—the only way to achieve the equivalent feature is to use the .NET Framework classes. Be aware that, even if the old way of doing things is still available, it's likely that it may go away in a future version of VB.

Features That Have .NET Framework Equivalents but Are Still There

There are a number of VB programming elements that are now supported by the .NET Framework but are also still supported in the VB .NET language. These include

- **String Manipulation functions such as Mid, Len, Trim, and InStr.** You should instead use the .NET Framework methods/properties: SubString, Length, Trim, and IndexOf, respectively.
- **Conversion functions such as CInt, CDbl, and CStr.** You should instead use the following .NET Framework methods of the Convert class:ToInt16 (or ToInt32), ToDouble, and ToString, respectively.
- **Math functions such as Rnd and Round.** These features are now supported by the .NET Framework methods, Random.NextDouble and Math.Round, respectively.
- **VB constants such as vbCrLf, vbTab, and vbRed.** These constants are now replaced with VB .NET's CrLf, Tab, and Red constants, respectively.
- **Date and time functions such as DateAdd, DateDiff, DateSerial, and TimeSerial.** These functions are now supported by various properties and methods of the .NET Framework's DateTime class.
- **String formatting functions such as Format, FormatCurrency, and FormatDateTime.** These functions are now supported by the .NET Framework ToString methods.

Try to reserve using the old VB6 style features for the conversion of legacy code and start using the .NET Framework version of these features as soon as possible.

NOTE

See Chapter 5, "Working with Numbers, Strings, Dates and Arrays in Visual Basic .NET" for more details on using the methods and the properties of the .NET Framework.

Features That Are Gone

A number of older, less useful VB features have been eliminated from the language in VB .NET. These include `DefType`, `GoSub`, `Let`, `Line`, `Circle`, `Pset`, `Scale`, `ERL`, and `static` procedures. It's doubtful that many ASP.NET developers will miss any of these antiquated features.

Summary

VB .NET is an exciting new version of Visual Basic. In this appendix, you learned about the key features that have changed in this version, and what to look out for when moving to VB .NET from VBScript or VB6.



INDEX

A

<a> (Anchor) HTML Server Control, 131
<a> tag, 363
Abs method (System.Math class), 72
accessibility of inherited properties and methods, 63
AccessKey property, 150
ACTION attribute (HTTP-POST form), 364
Active Server Method Extension (ASMX) file. *See ASMX files*
ADO (ActiveX Data Objects), 245
ADO.NET, 245

- DataReader objects, 274-276
- DataRow collection, 287
- DataSet objects, 283
 - creating, 284
 - creating from XML, 300-302
- DataRelation objects, 293
- fabricating, 295-299
- generating XML, 306, 310

- synchronizing updates, 322-324
- updating data, 310-311
- writing updates back to the database, 312, 318-319

DataTable objects, 284

- binding to DropDownList controls, 286
- manipulating rows, 287
- retrieving rows, 287

dichotomy, 246
namespaces, 246-247
.NET Data Providers, 258

- classes, 259
- OLE DB, 260
- SQL Server, 259

object model, 247
retrieving rows from a database, 271-272

AllowPaging attribute (DataGrid control), 207

AllowSorting attribute (DataGrid control), 205

Anchor (<a>) HTML Server Control, 131

application-level tracing, 377-378

applications

copying to the Web server, 417-418

.NET

IL, 19

requirements, 11

optimizing with caching services, 386

data caching, 390

page output caching, 387

user control output caching, 388

AppSettings section

MACHINE.CONFIG file, 412

WEB.CONFIG files, 413-414

ArrayLists, returning, 347

arrays

data binding, 250

multidimensional, 47

reversing element order, 93

searching, 94

sorting, 92

System.Array class. *See* System.Array class

VB .NET, 47, 429

checking upper bound, 49

ReDim/ReDim Preserve, 48

ASC keyword, 290

ASCX file extension, 231

ASMX (Active Server Method Extension) files, 334

applying WebMethod values, 346

importing class libraries, 342

returning

ArrayLists, 347

complex DataSets, 351

ASMX templates, 342

asp: tag prefix, 150

ASP.NET

ADO.NET. *See* ADO.NET

advantages, 358

ASMX file format, 335

asp: tag prefix, 150

automatic restarts, 36

caching services, 386

data caching, 390

page output caching, 387

user control output caching, 388

CheckBoxList control, 188

CLR Debugger, 379-380

breakpoints, 384

debugging ASP.NET pages, 381, 384

debugging compiled components, 385

coding ASP.NET clients using a SOAP proxy, 367-368

configuring runtime behavior, 34

consuming Web Services, 357-358

<message> elements, 361

<PortType> section, 360

<schema> elements, 362

services and bindings, 359

data and output caching, 35

data binding, 248

arrays and collection classes, 250

at runtime, 249

complex list controls to the

DataSet class, 251-254, 257

properties, methods, and functions, 248

DataTable objects, 289

DataView objects

compared to SQL View objects, 289

creating custom views, 289-292

deploying and updating solutions, 34

flexibility, 10

- HTTP Web Service clients, 362-364
infrastructure, 33
NET. platform, 12
 device independent, 13
 Internet-based component services, 14
 language support, 13-14
 OS-neutral environment, 12
overview, 9
publishing Web Services, 341
 ASMX templates, 342
 building public WebMethods, 343
 default SDLHelpGenerator pages, 344
 returning ArrayLists, 347
 returning complex DataSets, 351
 returning custom classes, 348
 WebMethod attribute, 346
 WebService attribute, 345
security, 33, 393
 authentication, 394
 authorization, 394
 FBS (Forms-Based Security), 402-406
 forms authentication, 397
 impersonation, 395
 ISS, 396-397
 WBS (Windows-based Security), 398-401
session state, 35
SOAP Web Service clients
 creating, 364
 generating SOAP proxies with WSDL.EXE, 365-367
solutions
 accessing configuration data at runtime, 414
 configuration elements, 412
 configuration inheritance, 411
 configuring, 410
copying applications to the Web server, 417-418
deploying, 415
deployment model, 416
MACHINE.CONFIG file, 410
registration, versioning, and replacement, 423
virtual databases, 419-421
WEB.CONFIG file configuration elements, 413
tracing services, 374
 application-level, 377-378
 customizing output, 376
 default output, 375
 page-level, 374
uploading files, 146
user controls, 231
 adding custom properties to, 234-235
 creating, 231
 enhanced features, 237
 event handling, 237-238
 loading dynamically, 240
 markup-only, 232
 WSDL.EXE, 335
- ASP.NET CONFIG files, 412**
- ASP.NET Web Forms**
- declarative input validation controls, 28
 - multiple client support, 26
 - programming model, 26
 - separating code from content, 26
 - Web Controls, 28
 - Postback feature, 29
 - templated controls, 31
 - ViewState feature, 29
 - Web Services, 32
 - multiple protocol support, 33
 - programming model, 32

ASP.NET Web Forms model, 109-110

automatic state management, 123-125
client-side events supported by server-side code, 115
event handling, 116
 accessing the event arguments, 119
 AutoPostBack property, 121
 defining events, 116
 Object arguments, 117
 responding to events, 117

HTML Server Controls, 127

Anchor (<a>), 131
division (div), 134
file input, 145
form, 133
general controls, 131
HTMLControl class, 130
Image (IMG), 132
runat="server" attribute, 128-129
select, 142, 144
span, 134
submit, reset, image, and button, 140
table controls, 135
table detail (td), 136
table header (th), 136
table row (tr), 136
text, password, textarea, and hidden, 138

in-page versus code-behind format, 110-113

object life cycle, 114-115
round-tripping, 116

assemblies, 68, 97

creating within a namespace definition, 100
namespaces, 98
 compiling, 102
 compiling with imported namespaces, 105

creating, 99
importing, 102-103

assignment operators (VB .NET), 430

Attributes property, 150

authentication, 394

FBS, 403
 implementing, 406
 storing user credentials, 404
 user login forms, 405
Integrated Windows (NTLM), 400
WBS, 399-400

authoring Web pages, 25-26

authorization, 394

FBS, 406
WBS, 401

automatic state management (ASP.NET Web Forms), 123, 125

AutoPostBack property, 121, 156-158

B

BackColor property, 150

base classes, 58

BaseValidator class, 216

BIN folders, 416

BinarySearch method (System.Array class), 94

binding

arrays and collections, 250
complex list controls to the DataSet class, 251-254, 257
data collections to CheckBoxList controls, 190
DataReader object to DataGrid controls, 274, 276
DataTable objects to DropDownList controls, 286
 properties, methods, and functions, 248

<Binding> sections, 360

bindings information type (WSDL documents), 358

BorderColor property, 150

BorderStyle property, 150

BorderWidth property, 150

branching (VB .NET), 52

- If...Then...Else statement, 52

- Select...Case statement, 53

breakpoints, 384

BufferResponse value, 346

button HTML Server Control, 140

Button Web server control, 168

- listing, 168

- OnCommand Attribute, 169

buttons

- reset, 141

- responding to button click events, 142

ByRef parameters, 428

ByVal parameters, 428

C

CacheDuration method, 346

caching services, 386

- data caching, 390

- page output caching, 387

- user control output caching, 388

calling

- methods, 343

- subroutines, 43

Ceiling method (System.Math class), 70

CheckBox Web server control, 162-163

CheckBoxList controls, 188

- binding data collections to, 190

- data-bound, 191

- listing, 189

checkedChanged event (RadioButton Web server control), 160

class attribute, 342

class constructors, 65

classes, 57

- adding methods, 343

- argument objects, 119

- creating (VB .NET), 58

- custom, returning, 348

- data, 22

- drawing, 23

- inheritance, 58

- libraries, 17, 21

- members, 68

- namespaces, 68

- .NET Data Providers, 259

- polymorphism, 58

- proxy (SOAP), 366

- publishing, 334

- system, 21

- System.Array, 91

- determining boundaries, 91

- Reverse method, 93

- searching arrays, 94

- Sort method, 92

- System.Convert, 84-86

- System.DateTime, 79

- adding/subtracting Date/Time values, 80

- Compare method, 80

- creating values, 80

- Now and Today properties, 79

- parsing DateTime values, 83

- Subtract method, 82

- System.Math, 69

- Abs method, 72

- calculating powers, roots, and logarithms, 70

- rounding and truncating numbers, 69

- Sign method, 72

- Sin method, 69

System.String, 72
 changing case of strings, 79
 CompareOrdinal method, 79
 comparing strings, 78
 EndsWith method, 75
 IndexOf method, 74
 Replace method, 75
 searching for and extracting substrings, 73
 splitting and joining strings, 76
 StartsWith method, 75
 Substring method, 75
 Trim method, 77
 trimming and padding strings, 76
 TrimStart method, 77

Web, 23
Windows Forms, 23
XML, 22

classname attribute, 232

client-side events, supporting with server-side code, 115

CLR (Common Language Runtime), 11, 17
 code management and execution, 18-19
 error handling, 20
 garbage collection, 20
 security support, 20
 services, 18

CLR Debugger, 379

code, 331, 342. *See also* listings

code management service (CLR), 18-19

code-behind coding, 112-113

collections
 creating, 56
 data binding, 250
 DataRows, 287

Columns property (TextBox Web server control), 156

command button-style controls, 169

Command classes, 247

command line options
 WSDL.EXE, 366
 XCOPY, 418

Command objects, executing SQL statements against data sources, 261-264
 retrieving a single value, 265-266
 stored procedure parameters, 267, 270-271

comments (VB .NET), 41

Compare method
 System.DateTime class, 80
 System.String class, 72, 78

CompareOrdinal method (System.String class), 79

CompareValidator controls, 219

comparing
 dates, 80
 strings, 78

compiling
 errors (Property statements), 61
 namespace assemblies, 102

complex DataSets
 requesting from Web methods, 352
 returning, 351

components, debugging, 385

Concat method (System.String class), 76

ConfigSections section (MACHINE.CONFIG file), 412

configuration elements (WEB.CONFIG files), 413

configuration files (ASP.NET), 410-411

configuring
 ASP.NET solutions, 410
 accessing configuration data at runtime, 414
 configuration elements, 412
 configuration inheritance, 411
 configuration model, 410
 MACHINE.CONFIG file, 410
 runtime behavior, 34
 WEB.CONFIG file configuration elements, 413

- IIS for authentication, 400
 - Web solutions, 409
 - Connection classes, 247, 260**
 - constants (VB .NET), 46**
 - constructing objects, 65**
 - consuming**
 - SOAP messages, 334
 - Web Services, 335-359
 - contracts (WSDL), 357**
 - control directive, 232**
 - controls**
 - hiding, 154
 - HTML Server Controls.* *See* *HTML Server Controls*
 - List, 181
 - CheckBoxList, 188-191
 - DataGridView control, 203-207, 210
 - DataList control, 198-199
 - RadioButtonList, 182-185
 - Repeater control, 195
 - templated, 194
 - Panel, 155
 - user. *See* user controls
 - validation. *See* validation controls
 - ControlToValidate property, 216**
 - converting values from one data type to another, 84**
 - copying applications to the Web server, 417-418**
 - Count property (DataRow collection), 287**
 - CreateDataSet function, 278**
 - CreateDataSource() method, 185**
 - creating Web pages, 25-26**
 - CssClass property, 150**
 - custom classes, returning, 348**
 - custom formatting**
 - DateTime values, 89
 - numbers, 87
 - custom properties, adding to user controls, 234**
 - customErrors section (WEB.CONFIG files), 414**
 - CustomValidator controls, 223**
- D**
- data**
 - binding (ASP.NET), 248
 - arrays and collection classes, 250
 - at runtime, 249
 - complex list controls to the DataSet class, 251-254, 257
 - properties, methods, and functions, 248
 - caching, 387, 390
 - filtering, 290
 - validation controls, 215
 - data classes, 17, 22, 247**
 - data sources**
 - Connection classes, 260
 - executing SQL statements against, 261-264
 - retrieving a single value, 265-266
 - stored procedure parameters, 267, 270-271
 - data types (VB .NET), 44-45**
 - data-bound RadioButtonList controls, 185**
 - DataAdapter classes, 247**
 - DataAdapter objects, creating DataSet objects, 276-279**
 - Database classes (ADO.NET), 247**
 - databases**
 - retrieving rows from, 271-272
 - SQL, 289
 - updating with custom SQL statements, 321
 - virtual
 - creating on Web servers, 419
 - Internet Information Server Manager, 420
 - WSH script, 421

- writing DataSet object updates back to database, 312, 318-319
 - CommandBuilder generated SQL statements, 312
 - listing, 312
 - synchronizing updates, 322-324
- DataBind method, 248**
- DataGrid controls, 203**
 - binding to DataReader objects, 274-276
 - editing
 - columns, 210
 - data rows in place, 320
 - listing, 203
 - paging, 207
 - sorting contents, 205
- DataGrid page element (EventArgs object), 120**
- DataGrid server controls, binding to a DataView object, 251**
- DataList controls, 198**
 - data binding, 254, 257
 - listing, 199
 - templated, 31
 - templates supported, 199
- DataReader objects, 271**
 - ExecuteReader method, 272
 - Read method, 272
 - using with DataGrid controls, 274-276
- DataRelation objects, 293**
- DataRow collection, 287**
- DataRowVersion enumeration values, 324**
- DataSet class**
 - ADO.NET System.Data namespace, 247
 - binding to complex list controls, 251-254, 257
- DataSet objects, 271, 283**
 - complex
 - requesting from Web methods, 352
 - returning, 351
- creating, 284
 - with DataAdapter objects, 276-279
 - from XML, 300-302
- DataRelation objects, 293**
- DataTable objects, 284**
 - binding to DropDownList controls, 286
 - deleting rows, 310
 - manipulating rows, 287
 - retrieving rows, 287
 - fabricating, 295-296
 - adding rows, 299
 - creating columns, 298
 - creating PrimaryKey constraint, 299
 - generating XML from, 306, 310
 - ReadXml method, 300
 - updating data, 310
 - managing the changed status, 311
 - synchronizing updates, 322-324
 - writing updates back to the database, 312, 318-319
- DataTable objects, 284**
 - binding to DropDownList controls, 286
 - DataView objects, 289-292
 - DefaultView property, 289
 - deleting rows, 310
 - manipulating rows, 287
 - PrimaryKey constraint, 299
 - retrieving rows, 287
- DataView objects, 289**
 - binding to a DataGrid server control, 251
 - custom views, 289-292
- DateTime values**
 - adding and subtracting, 80
 - comparing, 80

- formatting, 89-91
 - parsing, 83
 - DbCommit subroutine, 318**
 - debuggers (CLR Debugger), 379-380**
 - breakpoints, 384
 - debugging ASP.NET pages, 381, 384
 - debugging compiled components, 385
 - declaring variables, 44**
 - implicit and explicit declarations, 46
 - Option Explicit declarations, 47
 - Option Strict declarations, 47
 - default SDLHelpGenerator pages, 344**
 - DefaultView property (DataTable objects), 289**
 - defining events (ASP.NET Web Forms), 116**
 - deploying**
 - ASP.NET solutions, 415
 - copying applications to the Web server, 417-418
 - deployment model, 416
 - registration, versioning, and replacement, 423
 - virtual databases, 419-421
 - Web solutions, 409
 - derived classes, 58**
 - Description attribute, 342**
 - Description value, 346**
 - dichotomy (ADO.NET), 246**
 - Dim, 430**
 - Display property (validation controls), 216**
 - <div> HTML control, 134, 156**
 - Do...Loop statement, 53-54**
 - documents (WSDL), 357**
 - formats, 358
 - <message> elements, 361
 - <PortType> section, 360
 - <schema> elements, 362
 - services and bindings, 359
 - drawing classes, 17, 23**
 - DropDownList Web server control, 163**
 - binding to DataTable objects, 286
 - populating, 166
 - retrieving selected ListItem control, 164
 - DsFab DataSet, 298**
-
- ## E
- E field (System.Math class), 69**
 - editing columns (DataGrid controls), 210**
 - elements (WSDL documents), 360**
 - Enabled property, 150**
 - EnableSession value, 346**
 - EnableViewState="false" attribute, 125**
 - EndsWith method (System.String class), 75**
 - enumerations (NumberStyles), 85**
 - error handling, 426**
 - CLR, 18-20
 - VB versus VB .NET, 429
 - ErrorMessage property (validation controls), 217**
 - event handling**
 - ASP.NET Web Forms, 116
 - accessing the event arguments, 119
 - AutoPostBack property, 121
 - defining events, 116
 - Object arguments, 117
 - responding to events, 117
 - user controls, 237-238
 - EventArgs class, 119**
 - events**
 - AutoPostBack, 122
 - defining, 116
 - Object arguments, 117
 - round-tripping, 116
 - executable methods, publishing via the Internet, 329**

ExecuteNonQuery method, 261
ExecuteReader method (DataReader object), 272
ExecuteScalar method, 265
executing Web methods, 344
Exp method (System.Math class), 71
explicit variable declarations, 46
extracting substrings, 73

F

FabricateDataSet function, 298
FBS (Form-Based Security), 402
 advantage, 402
 authentication, 403
 implementing, 406
 storing user credentials, 404
 user login forms, 405
 authorization, 406
FCL (Framework class libraries), 21
 data classes, 22
 drawing classes, 23
 system classes, 21
 Web classes, 23
 Windows Forms classes, 23
 XML classes, 22
fields, 68-69
File Explorer, creating virtual databases on Web servers, 419
file input HTML Server Control, 145
FillList subroutine, 54
Floor method (System.Math class), 70
folders, controlling access to, 402
Font property, 150
For...Each statement, 56-57
For...Next statement, 55
ForeColor property, 150
form HTML Server Control, 133
form post backs, 151

form Web server controls, 156
 Button, 168
 CheckBox, 162
 DropDownList, 163, 166
 ImageButton, 170-174
 ListBox, 166, 168
 post away forms, 174
 RadioButton, 159, 162
 TextBox, 156-158
formatting
 DateTime values, 89-91
 numbers, 87
 values, 86
forms
 post away, 174
 reset button, 141
 user login, 404
 validation controls, 216
 advanced, 221
 error messages, 217
 properties and methods, 216
Forms-Based Security. *See FBS*
<forms> tag, 404
Framework class libraries. *See FCL*
Friend keyword, 64
FTP (File Transfer Protocol), 418
functions
 binding, 248
 string manipulation, 431
 VB .NET, 43

G

garbage collection service (CLR), 18, 20
general controls (HTML Server Controls), 131
 Anchor (<a>), 131
 division (div), 134
 form, 133

Image (IMG), 132
 span, 134

general Web server controls, 151

- HyperLink, 151
- Image, 153
- Label, 154
- LinkButton, 151
- Panel, 154-155

generating

- SOAP proxies with WSDL.EXE, 365-367
- tables, 177

GetLowerBound method (System.Array class), 91

GetUpperBound method (System.Array class), 91

GetXml method (DataSet object), 306

GetXmlSchema method (DataSet object), 306

input controls
 file input, 145
 select, 142-144
 submit, reset, image, and button, 140
 text, password, textarea, and hidden, 138
 runat="server" attribute, 128-129
 table controls, 135-136

HTMLControl class, 130

HTTP input message element, 361

HTTP Web Service clients, 362-364

HTTP-GET messages, 363

HTTP-POST messages, 363-364

HyperLink Web server control, 151

hyperlinks, creating, 151

H

handling events

- ASP.NET Web Forms, 116-121
- user controls, 237

Height property, 150

hidden HTML Server Control, 138

hiding controls, 154

HTML tags, 129

HTML POST FORM, calling Web Services, 336

HTML Server Controls, 127

- general controls, 131
 - Anchor (<a>), 131
 - division (div), 134
 - form, 133
 - Image (IMG), 132
 - span, 134
- HTMLControl class, 130

I

IComparer interface, searching arrays, 94

If...Then...Else statement, 52

IIS, 396-397

- IUSR_<machinename>, 395
- role of, 396
- Windows authentication, 401

IIS Manager, 400

IL (Intermediate Language) files, 18

Image (IMG) HTML Server Control, 132, 140

Image Web server control, 153

ImageAlign property (tag), 153

ImageButton Web server control, 170

- listing, 171
- OnCommand attribute, 172
- passing information to OnCommand event handler, 174

** tag, 153**

impersonation, 395

implementing ASP.NET security, 393

- authentication, 394
- authorization, 394

- FBS (Forms-Based Security), 402-406
 - forms authentication, 397
 - impersonation, 395
 - interaction with IIS, 396-397
 - WBS (Windows-based Security), 398-401
 - implicit variable declarations, 46**
 - import statements, 342**
 - importing namespace assemblies, 102-103**
 - in-page coding, 110**
 - IndexOf method**
 - System.Array class, 94
 - System.String class, 73-74
 - IndexOfAny method (System.String class), 73**
 - inheritance, 58**
 - accessibility of inherited properties and methods, 63
 - configuration, 411
 - Inherits statements (VB .NET), 62-63**
 - InnerHTML attribute (<td> tag), 129**
 - input controls**
 - file input, 145
 - select, 142-144
 - submit, reset, image, and button, 140
 - text, password, textarea, and hidden, 138
 - instance members, 68**
 - Integrated Windows (NTLM) authentication, 400**
 - Internet, 10**
 - Internet Explorer, 401**
 - Internet Information Server Manager, 420**
 - Internet-based component services, 14**
 - IP addresses, IIS functionality, 396**
 - Item property (DataRow collection), 287**
 - IUSR_<machinename>, 395**
- J - L**
- Join method (System.String class), 76**
 - joining strings, 76**
 - /I command line option (WSDL.EXE), 366**
 - Label Web server control, 154**
 - Language attributes, 342**
 - languages**
 - .NET support, 13
 - MS .NET languages, 13
 - non-MS .NET languages, 14
 - VB, 40
 - features changing in move to VB .NET, 431
 - legacy issues, 427
 - VB .NET, 40
 - arrays, 47-49
 - branching, 52-53
 - comments, 41
 - constants, 46
 - constructing an object, 65
 - creating classes, 58
 - creating objects, 57
 - data types, 44-45
 - functions, 43
 - implicit and explicit variable declarations, 46
 - inheritance, 62-63
 - looping, 53-57
 - multidimensional arrays, 47
 - operators, 42-43
 - Option Explicit declarations, 47
 - Option Strict declarations, 47
 - optional parameters, 50
 - overriding methods, 64
 - parameters, 44
 - passing parameters, 49-50
 - Property statements, 59-61

- subroutines, 43
- variables, 44
- WSDL, 330
- LastIndexOf method (System.Array class), 94**
- length property (System.String class), 73**
- libraries, 17, 21**
- LinkButton Web server control, 151-152**
- List controls, 181**
 - CheckBoxList controls, 188
 - binding data collections to, 190
 - data-bound, 191
 - listing, 189
 - RadioButtonList control, 182
 - data-bound, 185
 - listing, 182
 - testing, 184
 - templated, 194
 - DataGrid, 203-207, 210
 - DataList, 198-199
 - Repeater List, 195
- ListBox Web server control, 166-168**
 - listing, 167-169
 - SelectionMode property, 166
- listings**
 - arrays
 - searching, 94-95
 - sorting, 92
 - ASMX files
 - publishing Web Services, 334
 - returning complex DataSets, 351
 - ASP.NET configuration file, 410-411
 - ASP.NET Web Forms
 - automatic state management, 124
 - code-behind coding, 113
 - in-page coding, 110
 - assemblies
 - creating within a namespace definition, 100
 - importing from ASP.NET, 104
 - authentication
 - HTML markup for user login forms, 404
 - implementing FBS user authentication, 405
 - user credential data stored in XML files, 404
 - Calc subroutine, 69
 - CheckBoxList controls, 189
 - data-bound, 191
 - data binding, 248
 - controls to an array, 250
 - DataGrid server control to a DataView object, 251
 - DataSet objects
 - appending XML rows to, 302
 - creating from XML, 300
 - DataTable objects, 284
 - fabricating, 296
 - generating XML, 306
 - synchronizing updates, 322
 - writing updates back to the database, 312
 - DataTable objects, 287
 - DataView objects, 290
 - DateTime values
 - adding/subtracting, 80-81
 - formatting, 90-91
 - parsing, 83

DoCount subroutine, 266
event handling (AutoPostBack event), 122
EventArgs object of the DataGrid page element, 120
ExecuteScalar method, 265
FBS for an ASP.NET application, 403
formatting numbers, 87
GEOMETRY.VB, 100
HTML POST FORM, calling Web Services, 336
HTML Server Controls
 accessing InnerHtml attribute of a <td> tag, 129
 Anchor (<a>) control, 131
 Div control, 134
 file input control, 146
 form control, 133
 Image (IMG) control, 132
 input controls, 138, 141
 login form, 128
 radio buttons and checkboxes, 144
 reset buttons, 141
 responding to button click events, 142
 select control, 142
 server-side code to set table attributes, 135
 setting browser window title, 129
 th, tr, and td tags, 136
IndexOf and IndexOfAny methods, 73
Object argument of server-side event handlers, 117
OleDbDataReader objects, reading records from a DataReader, 272
page-output caching, 388
parameterized stored procedures, 267
post away ASP.NET form, 175
Postback feature, 29
publishing Web Services, 334
RadioButtonList control, 182
 data-bound, 185
Repeater List control, 196
rounding and truncation methods, 70
secured SOAP Web Services, 337
SOAP
 request messages, 332
 response messages, 333
SQL statements, executing against data sources, 262
SqlDataAdapter objects, 277
SqlDataReader objects, binding to DataGrid control, 274
tables, generating programmatically, 177-178
templated DataList control, 31
TimeSpan values, 82
trace output, customizing, 375
user controls
 adding custom properties to, 234
 event handling, 237-238
 loading dynamically, 240
 markup-only, 232-233
 output caching, 389
 setting custom properties, 235
validation controls
 CompareValidator control, 219
 CustomValidator control, 223
 RangeValidator control, 220
 RegularExpressionValidator control, 222
 RequiredFieldValidator control, 218
 ValidationSummary control, 229
ViewState feature, 29
Web server controls
 Button, 168-169
 CheckBox, 162-163
 DropDownList, 165
 HyperLink, 151
 Image, 153
 ImageButton, 171
 LinkButton, 152

- ListBox, 167-169
 - Panel, 154
 - RadioButton, 159
 - TextBox, 157-158
 - Web Services**
 - calling with ASP.NET Web forms, 368
 - returning ArrayLists, 347
 - returning custom classes, 348
 - sample ASP.NET Web Service, 358
 - session state support, 338
 - SOAP proxy class, 365
 - supporting transactions, 339
 - WEB.CONFIG files, appSettings section, 414
 - Windows authentication, testing, 399
 - WSDL documents**
 - bindings entries, 359
 - message elements, 361
 - PortType section, 360
 - schema elements, 362
 - service section, 359
 - WSDL message for SOAP Services, 331
 - WSH script, 421
 - loading user controls dynamically**, 240
 - log method (System.Math class)**, 71
 - logarithms**, 70
 - looping (VB .NET)**
 - Do...Loop statement, 53-54
 - For...Each statement, 56-57
 - For...Next statement, 55
 - While...End statement, 55
- M**
- MACHINE.CONFIG file**, 410, 412
 - markup-only user controls**
 - creating, 232
 - listing, 233
- MaxLength property (TextBox Web server control)**, 156
 - members (classes)**, 68
 - <message> elements (WSDL documents)**, 361
 - MessageName value**, 346
 - messages**
 - HTTP-GET, 363
 - HTTP-POST, 363-364
 - SOAP
 - publishing and consuming, 334
 - request, 332
 - response, 333
 - WSDL, 332
 - messages information type (WSDL documents)**, 358
 - methods**
 - adding to classes, 343
 - binding, 248
 - help pages, 344
 - HTMLControl class, 130
 - overloading, 428
 - overriding, 64
 - validation controls, 216
 - MIT SDK (Mobile Internet Tools)**, 13
 - MsgBox function**, 40
- N**
- /n command line option (WSDL.EXE)**, 366
 - namespaces**, 97
 - ADO.NET, 246-247
 - assemblies, 98
 - compiling, 102
 - compiling with imported namespaces, 105
 - creating within a namespace definition, 100
 - importing, 102-103
 - classes, 68

- creating namespace assemblies, 99
- defining namespace area, 99
- .NET, 12**
 - applications, 19
 - device independent, 13
 - Internet-based component services, 14
 - language support
 - MS .NET languages, 13
 - non-MS .NET languages, 14
 - operating system components, 11
 - OS-neutral environment, 12
 - platform utilities, 12
 - runtimes, 11
- .NET Data Providers, 258**
 - classes, 259
 - OLE DB, 260
 - SQL Server, 259
- .NET Framework**
 - ADO.NET, 245, 258
 - assemblies, 97
 - classes, 68
- .NET Platform**
 - CLR. *See* CLR
 - FCL (Framework class libraries), 21
- /nologo command line option (WSDL.EXE), 366**
- Now property (System.DateTime class), 79**
- NTLM, limitations, 401**
- numbers**
 - formatting, 86-87
 - powers, roots, and logarithms, 70
 - rounding and truncating, 69
- NumberStyles enumerations, 85**
- O**
- /o command line option (WSDL.EXE), 366**
- object models (ADO.NET), 247**
- object-oriented programming (VB .NET features), 426**
- objects**
 - classes, 57-59
 - constructing, 65
 - creating (VB .NET), 57
- OLE DB data provider, 258-260**
- OleDbCommand objects**
 - ExecuteNonQuery method, 261
 - ExecuteScalar method, 265
 - retrieving single values, 265-266
 - stored procedure parameters, 267, 270-271
- OleDbCommandBuilder objects, writing DataSet object updates back to the database, 312**
- OleDbConnection class, 260**
- OleDbDataAdapter objects, 276-279**
- OleDbDataReader objects, 272**
- OnCommand attribute**
 - Button Web server control, 169
 - ImageButton Web server control, 172
- OnPageIndexChanged event (DataGridView control), 207**
- OnServerValidate method (CustomValidator control), 223**
- OnSortCommand attribute (DataGridView control), 205**
- OOP (object-oriented programming), 57, 428**
- operating system components, 11**
- operators (VB .NET), 42-43**
- optimizing applications with caching services, 386**
- Option Explicit, 426**
- Option Explicit declarations (VB .NET), 47**
- Option Strict declarations (VB .NET), 47**
- optional parameters (VB .NET), 50**
- OS-neutral environment, 12**
- output caching, 387-388**

OutputXML subroutine, 309
overloading methods and properties, 428
overriding methods, 64

P

padding strings, 76
page-level tracing, 374
Page_Load event, 115
Page_Unload event, 115
pages
 debugging, 381
 output caching, 387
paging (DataGrid controls), 207
Panel Web server control, 154-155
parameterized constructors, 65
parameters
 optional, 50
 passing
 by value vs. by reference, 50
 VB .NET, 49
 VB .NET, 44
parsing DateTime values, 83
passing parameters
 by value vs. by reference, 50
 VB .NET, 49
password HTML Server Control, 138
passwords (authentication), 394
/pd command line option (WSDL.EXE), 367
PI field (System.Math class), 69
polymorphism, 58
populating DropDownList Web server control, 166
port types information type (WSDL documents), 358
<PortType> section (WSDL documents), 360
post away forms, 174

Postback feature, 29
Pow method (System.Math class), 70
powers, 70
/pp command line option (WSDL.EXE), 367
PrimaryKey constraint (DataTable objects), 299
Private keyword, 63
programming
 code readability, 342
 OOP, 57
 Web, 109
 ASP.NET Web Form object life cycle, 114-115
 client-side events supported by server-side code, 115
 in-page versus code-behind format, 110-113
programming model (Web page authoring), 26
properties
 binding, 248
 HTMLControl class, 130
 overloading, 428
 validation controls, 216
 Web server controls, 150, 156
Property statements
 compile errors, 61
 VB .NET, 59-61
Protected Friend keyword, 64
Protected keyword, 63
/protocol command line option (WSDL.EXE), 367
protocols (SOAP), 330
proxies (SOAP)
 coding an ASP.NET client, 367-368
 generating with WSDL.EXE, 365-367
/pu command line option (WSDL.EXE), 367
public WebMethods, 343

public servers, publishing Web Services,
341

publishing

executable methods via the Internet,
329
SOAP messages, 334
Web Services, 334
 with ASP.NET, 341-342
 building public WebMethods,
 343
 default SDLHelpGenerator
 pages, 344
 listing, 334
 public servers, 341
 returning ArrayLists, 347
 returning complex DataSets, 351
 returning custom classes, 348
 WebMethod attribute, 346
 WebService attribute, 345

R

RadioButton Web server control, 159

binding a list of radio buttons to a data
source, 162
checking values of, 160
listing, 159

RadioButtonList control, 162, 182

data-bound, 185
testing, 184

RangeValidator controls, 220

Read method (DataReader object), 272

**ReadOnly property (TextBox Web server
control), 156**

readUserFile method, 406

ReadXml method (DataSet objects), 300

registering ASP.NET Web components,
423

RegularExpressionValidator controls, 221

Repeater List control

listings, 196
templates supported, 195

Replace method (System.String class), 75

replacing ASP.NET Web components, 423

RequiredFieldValidator controls, 217

reset buttons, 141

reset HTML Server Control, 140

retrieving Web Service data

HTTP-GET, 363
HTTP-POST, 363-364

returning

ArrayLists, 347
complex DataSets, 351
custom classes, 348

Reverse method (System.Array class), 93

root folders, BIN folders, 416

roots, 70

Round method (System.Math class), 69

round-tripping, 116

rounding numbers, 69

**rows, adding to fabricated DataSet
objects, 299**

**Rows property (TextBox Web server con-
trol), 156**

RPC (remote procedure calling) systems,
14

runat="server" attribute, 128-129

**runtime section (MACHINE.CONFIG
file), 412**

runtimes

accessing configuration data, 414
CLR, 17
 code management and execution,
 18-19
 error handling, 20
 garbage collection, 20
 security support, 20
services, 18

-
- schema information type (WSDL documents),** 358
- <schema> elements (WSDL documents),** 362
- <script> tag,** 111
- SDLHelpGenerator pages,** 344
- searching**
- arrays, 94
 - for substrings, 73
- secured sockets layers (SSL),** 337
- security, 393**
- ASP.NET, 33
 - authentication, 394
 - authorization, 394
 - FBS (Forms-Based Security), 402
 - authentication, 403-406
 - authorization, 406
 - forms authentication, 397
 - IIS, 396-397
 - impersonation, 395
 - WBS (Windows-based Security), 398
 - authentication, 399-400
 - authorization, 401
 - Web Services, 337
- security support service (CLR),** 18-20
- select HTML Server Control,** 142-144
- Select...Case statement,** 53
- SelectionMode property, setting from an HTML attribute,** 166
- server-side event handling,** 117, 121
- server-side script, setting custom control properties,** 236
- /server command line option (WSDL.EXE),** 366
- servers**
- controlling access to resources, 402
 - deploying Webs to, 416
 - updating compiled components, 416
- <service> section (WSDL documents),** 359
- services information type (WSDL documents),** 358
- session state (ASP.NET),** 35
- sessionState section (WEB.CONFIG files),** 413
- Set keyword,** 426
- setting custom user control properties,** 236
- Sign method (System.Math class),** 72
- Simple Object Access Protocol.** *See SOAP*
- Sin method,** 69
- smart server-side controls,** 110
- SOAP (Simple Object Access Protocol),** 14
- calling methods, 343
 - coding ASP.NET clients with a SOAP proxy, 367-368
 - messages
 - creating, 334
 - publishing and consuming, 334
 - request messages, 332
 - response messages, 333
 - Web Service clients
 - creating, 364
 - generating SOAP proxies with WSDL.EXE, 365-367
- SOAP protocol,** 330
- Sort method (System.Array class),** 92
- sorting DataGrid control contents,** 205
- SourceVersion property,** 324
- span HTML Server Control,** 134
- Split method (System.String class),** 76
- splitting strings,** 76
- SQL databases,** 289
- SQL Server data provider,** 258-260
- SQL statements**
- custom, 321
 - executing against data sources, 261-264
 - retrieving a single value, 265-266
 - stored procedure parameters, 267, 270-271

SqlCommand object

ExecuteNonQuery method, 261
ExecuteScalar method, 265
Parameters collection, 270
retrieving single values, 265-266
stored procedure parameters, 267,
270-271

SqlCommandBuilder objects, writing

DataSet object updates back to the database, 312

SqlConnection class, 260
SqlDataAdapter objects, 276-279
SqlDataReader objects, 274
SqlDbType enumeration values, 270
SqlParameter constructor, 270
Sqrt method (System.Math class), 71
SSL (secured sockets layers), 337
standalone debuggers (CLR Debugger),
379

StartsWith method (System.String class),
75, 79

state management

ASP.NET Web Forms, 123-125
Web Services, 338

statements

calling subroutines, 43
VB .NET, Inherits, 62

static members, 68, 72**static methods**

rounding and truncation, 69
System.Convert class, 84

stored procedures, parameters, 267**storing**

user credential information (authentication), 404
username/password pairs, 394

strings

changing case of, 79
coercing TimeSpan values into, 82
comparing, 78

converting to DateTime values, 80
DateTime formatting, 89-91
determining length, 72
formatting, 86
joining, 76
manipulation functions, 431
Number formatting, 87
splitting, 76
System.String class, 72

Style property, 150**submit HTML Server Control, 140****subroutines (VB .NET), 43****Substring method (System.String class),
75**

substrings, searching for and extracting,
73

**Subtract method (System.DateTime
class), 82-84****SWATTRIB.ASMX, 346****System class library, 342****system classes, 17, 21, 68****System.Array class, 91**

determining boundaries, 91
Reverse method, 93
searching arrays, 94
Sort method, 92

System.Convert class, 84-86**System.Data namespace (ADO.NET),
246-247****System.Data.Common namespace
(ADO.NET), 246****System.Data.OleDb namespace
(ADO.NET), 246-247****System.Data.SqlClient namespace
(ADO.NET), 246-247****System.Data.SqlTypes namespace
(ADO.NET), 246****System.DateTime class, 79**

adding/subtracting DateTime values, 80
Compare method, 80

- creating values, 80
 - Now and Today properties, 79
 - parsing DateTime values, 83
 - Subtract method, 82
 - system.diagnostics section**
 - (MACHINE.CONFIG file), 412
 - System.Globalization class, 85**
 - System.Math class, 69**
 - Abs method, 72
 - calculating powers, roots, and logarithms, 70
 - rounding and truncating numbers, 69
 - Sign method, 72
 - Sin method, 69
 - system.net section (MACHINE.CONFIG file), 412**
 - System.String class, 72**
 - changing case of strings, 79
 - CompareOrdinal method, 79
 - comparing strings, 78
 - EndsWith method, 75
 - IndexOf method, 74
 - length property, 73
 - Replace method, 75
 - searching for and extracting substrings, 73
 - splitting and joining strings, 76
 - StartsWith method, 75
 - Substring method, 75
 - Trim method, 77
 - trimming and padding strings, 76
 - TrimStart method, 77
 - system.Web section, 399, 412**
 - System.Web.Services class library, 342**
-
- T**
- TabIndex property, 150**
 - table controls, 135-136**
 - table detail (td) HTML Server Control, 136**
 - table header (th) HTML Server Control, 136**
 - <table> HTML control, 156
 - table HTML Server Controls, 135**
 - table row (tr) HTML Server Control, 136**
 - table Web server controls, 176-177**
 - tagname attribute, 233**
 - tags, asp: prefix, 150**
 - <td> tags, 129
 - templated controls, 31**
 - templated List controls, 194**
 - DataGrid control, 203
 - editing columns, 210
 - listing, 203
 - paging, 207
 - sorting contents, 205
 - DataList control, 198-199
 - Repeater List control
 - listing, 196
 - templates supported, 195
 - templates (ASMX), 342-343**
 - testing**
 - radioButtonList controls, 184
 - WBS authentication, 399
 - Web Services, 363
 - text HTML Server Control, 138**
 - Text property (TextBox Web server control), 156**
 - textarea HTML Server Control, 138**
 - TextBox Web server control, 156-157**
 - AutoPostBack property, 158
 - listing, 157-158
 - TextChanged event, 158**
 - TextMode property (TextBox Web server control), 156**
 - TimeSpan values, 82**

Today property (System.DateTime class), 79
ToLower method (System.String class), 79
ToolTip property, 150
ToString method, 86
 format characters, 86
 listing, 87
ToUpper method (System.String class), 79
Trace.Warn methods, 375
Trace.Write method, 375
tracing services, 374
 application-level, 377-378
 customizing output, 376
 default output, 375
 page-level, 374
TransactionOption attribute, 339, 346
transaction management, 339
trigonometry, 69
Trim method (System.String class), 77
TrimEnd method (System.String class), 77
trimming strings, 76
TrimStart method (System.String class), 77
troubleshooting (CLR Debugger), 379-380
 break points, 384
 debugging ASP.NET pages, 381, 384
 debugging compiled components, 385
truncating numbers, 69

U

UBound function, 49
updating
 compiled components, 416
 databases with custom SQL statements, 321
 DataSet object data, 310
 managing the changed status, 311
 synchronizing updates, 322-324
 writing updates back to the database, 312, 318-319

uploading files with ASP.NET, 146
user controls, 231
 adding custom properties to, 234-235
 creating, 231, 238
 enhanced features, 237
 event handling, 237-238
 loading dynamically, 240
 markup-only, 232
 output caching, 388
user interface controls (MIT SDK), 13
usernames (authentication), 394

V

validateUser method, 406
validation controls, 215
 advanced, 221
 CompareValidator control, 219
 CustomValidator control, 223
 error messages, 217
 methods, 216
 overview, 215
 properties, 216
 RangeValidator control, 220
 RegularExpressionValidator control, 221
 RequiredFieldValidator control, 217
 ValidationSummary control, 229

ValidationSummary controls, 229

values
 converting from one data type to another, 84
 formatting, 86

variables
 arrays, 47
 checking upper bound, 49
 multidimensional, 47
 ReDim/ReDim Preserve, 48
 declaring, 47
 VB .NET, 44-46

Variant data type, 427**VB .NET, 40, 67**

- arrays, 47, 429
 - checking upper bound, 49
 - multidimensional, 47
- ReDim/ReDim Preserve, 48
- assignment operators, 430
- branching, 52
 - If...Then...Else statement, 52
 - Select...Case statement, 53
- Collection object, 56
- comments, 41
- constants, 46
- creating
 - classes, 58
 - objects, 57
- data types, 44-45
- Dim, 430
- error handling, 429
- features changing in move from VB, 431
- features eliminated in move from VB, 432
- functions, 43
- inheritance, 62-63
- looping
 - Do...Loop statement, 53-54
 - For...Each statement, 56-57
 - For...Next statement, 55
 - While...End statement, 55
- migrating
 - from VB6, 426
 - from VBScript, 425
- namespaces, 97
 - assemblies, 98
 - compiling namespace assemblies, 102
 - compiling with imported namespaces, 105
- creating assemblies within a namespace definition, 100
- creating namespace assemblies, 99
- defining namespace area, 99
- importing namespace assemblies, 102-103
- new features, 425
- objects, constructing, 65
- operators, 42-43
- OOP features, 428
- Option Explicit declarations, 47
- Option Strict declarations, 47
- overcoming legacy issues, 427
- overriding methods, 64
- parameters, 44
 - optional, 50
 - passing, 49-50
- Property statements, 59-61
- statements and lines, 41
- subroutines, 43
- variables, 44-46

VB6

- error handling, 429
- migrating to VB .NET, 426

VBScript

- error handling, 426
- migrating to VB .NET, 425

views, 289**ViewState feature, 29****virtual databases, creating on Web servers, 419**

- File Explorer, 419
- Internet Information Server Manager, 420
- WSH script, 421

Visual Basic

- .NET syntax and structure, 39
- VB .NET. *See* VB .NET

Visual Basic .NET. See **VB .NET**

Visual Basic 6

- error handling, 429
- migrating to VB .NET, 426

VS .NET

- ASP.NET benefits, 358
- creating Web pages, 114

W

WBS (Windows-based Security), 398

- authentication, 399-400
- authorization, 401

Web applications, configuration inheritance, 412

Web classes, 17, 23

Web control class, 150

Web Controls, 28

- Postback feature, 29
- templated controls, 31
- ViewState feature, 29

Web Form controls, List controls, 181

- CheckBoxList control, 188-191
- DataGrid control, 203-207, 210
- DataList control, 198-199
- RadioButtonList control, 182-185
- Repeater List control, 195
- templated, 194

Web Forms. See **ASP.NET Web Forms**

Web methods

- classes, 342
- executing, 344
- requesting complex DataSets, 352

Web pages

- authoring, 25-26
- creating, 109
 - ASP.NET Web Form object life cycle, 114-115
 - client-side events supported by server-side code, 115

in-page versus code-behind format, 110-113

VS .NET, 114

Panel controls, 155

Web server controls, 149

- categories, 150
- form controls, 156
 - Button, 168
 - CheckBox, 162
 - DropDownList, 163, 166
 - ImageButton, 170-174
 - ListBox, 166-168
 - post away forms, 174
 - RadioButton, 159, 162
 - TextBox, 156-158

general controls, 151

HyperLink, 151

Image, 153

Label, 154

LinkButton, 151

Panel, 154-155

HTML server controls (comparison), 150

overview, 149

properties, 150

table controls, 176-177

WebControl class, 150

Web servers

copying applications to, 417-418

virtual databases

creating, 419

File Explorer, 419

Internet Information Server Manager, 420

WSH script, 421

Web Services, 32, 329

consuming, 335-359

history of, 329

-
- HTML POST FORM, 336
 - HTTP Web Service clients, 362-364
 - multiple protocol support, 33
 - programming model, 32
 - publishing
 - with ASP.NET, 341-342
 - building public WebMethods, 343
 - default SDLHelpGenerator pages, 344
 - WebMethod attribute, 346
 - WebService attribute, 345
 - publishing and consuming, 334
 - returning
 - ArrayLists, 347
 - complex DataSets, 351
 - custom classes, 348
 - secured SOAP, 337
 - security, 337
 - SOAP protocol, 330
 - SOAP request messages, 332
 - SOAP response messages, 333
 - SOAP Web Service clients
 - creating, 364
 - generating SOAP proxies with WSDL.EXE, 365-367
 - state management, 338
 - transaction management, 339
 - WSDL, 330-332
 - WEB.CONFIG files**
 - accessing configuration data at runtime, 414
 - appSettings section, 414
 - authorization, 406
 - <authorization> sections, 403
 - configuration elements, 413
 - customErrors section, 414
 - sessionState section, 413
 - WebMethod attribute, 335, 343, 346**
 - WebMethods (public), 343**
 - WebService attribute, 345, 359**
 - While...End statement, 55**
 - Width property, 150**
 - Windows .NET Component Updated, 11**
 - windows classes, 17
 - Windows Forms classes, 23**
 - Windows-based Security. *See* WBS**
 - Winer, David, 329**
 - Wrap property (TextBox Web server control), 156**
 - WriteXml method (DataSet object), 306, 310**
 - WriteXmlSchema method (DataSet object), 306**
 - writing DataSet object updates back to the database, 312, 318-319**
 - CommandBuilder generated SQL statements, 312
 - listing, 312
 - synchronizing updates, 322-324
 - WSDL (Web Service Description Language), 330**
 - contracts, 357
 - documents
 - format, 358
 - <message> elements, 361
 - <PortType> section, 360
 - <schema> elements, 362
 - services and bindings, 359
 - example message, 332
 - setting values for Web Services, 345
 - WSDL.EXE, 335**
 - command line options, 366
 - generating a SOAP proxy, 365-367
 - WSH script, creating virtual databases on Web servers, 421**

X-Y-Z

XCOPY script, 418

XML (eXtensible Markup Language)

 classes, 17, 22

 creating DataSet objects, 300-302

 generating from DataSet objects, 306,
 310

XML Web services, 11

XML-based configuration files, 410

XML-RPC, 329

XmlReadMode enumeration values, 302

XmlTextReader objects, 302