2ND EDITION

sitepoint

BUILD YOUR OWN ASP.NET 2.0 WEB SITE USING C# & VB

BY CRISTIAN DARIE
& ZAK RUVALCABA

THE ULTIMATE ASP.NET BEGINNER'S GUIDE

Build Your Own ASP.NET 2.0 Web Site Using C# & VB
(Chapters 1, 2, 3, and 4)

Thank you for downloading these four chapters of Cristian Darie and Zac Ruvalcaba's *Build Your Own ASP.NET Web Site Using C# & VB*.

This excerpt encapsulates the Summary of Contents, Information about the Author and SitePoint, Table of Contents, Preface, and the first four chapters of the book.

We hope you find this information useful in evaluating the book.

Summary of Contents of this Excerpt

Preface xi
1. Introduction to CSS 1
2. ASP.NET Basics 33
3. VB and C# Programming Basics 51
4. Constructing ASP.NET Web Pages 93
Index 659

Summary of Additional Book Contents

5. Building Web Applications 143
6. Using the Validation Controls 219
7. Database Design and Development 251
8. Speaking SQL 293
9. ADO.NET 331
10. Displaying Content Using Data Lists 401
11. Managing Content Using Grid View and Details View ... 427
12. Advanced Data Access 469
13. Security and User Authentication 527
14. Working with Files and Email 571
A. Web Control Reference 611

Build Your Own ASP.NET 2.0 Web Site Using C# & VB

by Cristian Darie

and Zak Ruvalcaba

Build Your Own ASP.NET 2.0 Web Site Using C# & VB

by Cristian Darie and Zak Ruvalcaba

Expert Reviewer: Wyatt Barnett
Expert Reviewer: Sara Smith
Managing Editor: Simon Mackie
Technical Editor: Craig Anderson
Technical Director: Kevin Yank
Printing History:
First Edition: April 2004
Second Edition: October 2006

Editor: Georgina Laidlaw
Index Editor: Max McMaster
Cover Design: Jess Mason
Cover Layout: Alex Walker

Notice of Rights

Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors, will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware products described herein.

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.

Published by SitePoint Pty. Ltd.

424 Smith Street Collingwood
VIC Australia 3066.
Web: www.sitepoint.com
Email: business@sitepoint.com

ISBN 0-9752402-8-5
Printed and bound in the United States of America

About the Authors

Zak Ruvalcaba has been designing, developing, and researching for the Web since 1995. He holds a Bachelor's Degree from San Diego State University and a Master of Science in Instructional Technology from National University in San Diego.

In the course of his career, Zak has developed web applications for such companies as Gateway, HP, Toshiba, and IBM. More recently, he's worked as a wireless software engineer developing .NET solutions for Goldman Sachs, TV Guide, The Gartner Group, Microsoft, and Qualcomm. Currently, Zak holds a programming position with ADCS Inc. in San Diego supporting internal .NET applications.

Previous books by Zak Ruvalcaba include *The 10 Minute Guide to Dreamweaver 4* (Que Publishing) and *Dreamweaver MX Unleashed* (Sams Publishing). He also lectures on various technologies and tools, including Dreamweaver and ASP.NET, for the San Diego Community College District.

Cristian Darie is a software engineer with experience in a wide range of modern technologies, and the author of numerous technical books, including the popular *Beginning E-Commerce* series. Having worked with computers since he was old enough to use a keyboard, he initially tasted programming success with a prize in his first programming contest at the age of 12. From there, Cristian moved on to many other similar achievements, and is now studying distributed application architectures for his PhD.

He always loves hearing feedback about his books, so don't hesitate to drop him a "hello" message when you have a spare moment. Cristian can be contacted through his personal web site at http://www.cristiandarie.ro.

About the Expert Reviewers

Wyatt Barnett leads the in-house development team for a major industry trade association in Washington DC. He also writes for SitePoint's .NET Blog, *The Daily Catch*.[1]

Sara Smith is an ASP.NET contractor for the US Army and is also a partner in a web development business, brainyminds. She has been working with the .NET framework since its early days. Sara just relocated to Belgium from the US with her family.

About the Technical Editor

Before joining SitePoint, Craig Anderson studied Computer Science at RMIT University, then worked as a web developer for five years. He spent much of this time trying to convince Visual Basic developers that one of these days they would have to learn object oriented programming.

[1] http://www.sitepoint.com/blogs/category/net/

Craig plays bass guitar in Melbourne rock band Look Who's Toxic,[2] and indulges in all the extracurricular activities you'd expect of a computer nerd/musician approaching 30 (other than role playing—somehow he never got into that).

About the Technical Director

As Technical Director for SitePoint, Kevin Yank oversees all of its technical publications—books, articles, newsletters, and blogs. He has written over 50 articles for SitePoint, but is best known for his book, *Build Your Own Database Driven Website Using PHP & MySQL*. Kevin lives in Melbourne, Australia, and enjoys performing improvised comedy theatre and flying light aircraft.

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit http://www.sitepoint.com/ to access our books, newsletters, articles, and community forums.

[2] http://www.lookwhostoxic.com/

For my wife Jessica.

—Zak Ruvalcaba

To my family and friends.

—Cristian Darie

Table of Contents

Preface ..... **xi**

1. Introducing ASP.NET and the .NET Platform **1**
- What is ASP.NET? 2
- Installing the Required Software 5
  - Installing the Web Server 6
  - Installing the .NET Framework and the SDK 9
  - Configuring the Web Server 11
  - Installing SQL Server 2005 Express Edition 22
  - Installing SQL Server Management Studio Express 22
  - Installing Visual Web Developer 2005 25
- Writing your First ASP.NET Page 26
- Getting Help 32
- Summary 32

2. ASP.NET Basics **33**
- ASP.NET Page Structure 34
  - Directives 36
  - Code Declaration Blocks 37
  - Code Render Blocks 39
  - ASP.NET Server Controls 40
  - Server-side Comments 41
  - Literal Text and HTML Tags 42
- View State 44
- Working with Directives 47
- ASP.NET Languages 48
  - Visual Basic 48
  - C# 49
- Summary 49

3. VB and C# Programming Basics **51**
- Programming Basics 51
  - Control Events and Subroutines 52
  - Page Events 56
  - Variables and Variable Declaration 59
  - Arrays 62
  - Functions 65
  - Operators 68
  - Conditional Logic 70
  - Loops 72

Object Oriented Programming Concepts 76
Objects and Classes 77
Properties 80
Methods 81
Classes 81
Constructors 81
Scope 82
Events 83
Understanding Inheritance 83
Objects In .NET 84
Namespaces 86
Using Code-behind Files 87
Summary 91

4. Constructing ASP.NET Web Pages 93
Web Forms 94
HTML Server Controls 95
Using the HTML Server Controls 97
Web Server Controls 101
Standard Web Server Controls 103
List Controls 110
Advanced Controls 112
Web User Controls 126
Creating a Web User Control 126
Master Pages 132
Using Cascading Style Sheets (CSS) 135
Types of Styles and Style Sheets 136
Summary 141

5. Building Web Applications 143
Introducing the Dorknozzle Project 144
Using Visual Web Developer 147
Meeting the Features 148
Executing your Project 156
Using Visual Web Developer's Built-in Web Server 157
Using IIS 160
Core Web Application Features 166
Web.config 166
Global.asax 170
Using Application State 173
Working with User Sessions 180
Using the `Cache` Object 182

Using Cookies 183
Starting the Dorknozzle Project 186
Preparing the Sitemap 187
Using Themes, Skins, and Styles 189
Building the Master Page 195
Using the Master Page 199
Extending Dorknozzle 201
Debugging and Error Handling 204
Debugging with Visual Web Developer 204
Other Kinds of Errors 210
Custom Errors 212
Handling Exceptions Locally 213
Summary 218

6. Using the Validation Controls 219
Introducing the ASP.NET Validation Controls 220
Enforcing Validation on the Server 223
Using Validation Controls 229
RequiredFieldValidator 230
CompareValidator 231
RangeValidator 233
ValidationSummary 235
RegularExpressionValidator 236
CustomValidator 239
Validation Groups 242
Updating Dorknozzle 245
Summary 250

7. Database Design and Development 251
What is a Database? 252
Creating your First Database 254
Creating a New Database Using Visual Web Developer 255
Creating a New Database Using SQL Server Management Studio 256
Creating Database Tables 258
Data Types 262
Column Properties 264
Primary Keys 265
Creating the Employees Table 267
Creating the Remaining Tables 271
Populating the Data Tables 273
Relational Database Design Concepts 276

Foreign Keys 278
Using Database Diagrams 280
Implementing Relationships in the Dorknozzle Database 284
Diagrams and Table Relationships 287
Summary 292

8. Speaking SQL 293
Reading Data from a Single Table 294
Using the `SELECT` Statement 297
Selecting Certain Fields 299
Selecting Unique Data with `DISTINCT` 300
Row Filtering with `WHERE` 302
Selecting Ranges of Values with `BETWEEN` 303
Matching Patterns with `LIKE` 304
Using the `IN` Operator 305
Sorting Results Using `ORDER BY` 306
Limiting the Number of Results with `TOP` 307
Reading Data from Multiple Tables 307
Subqueries 308
Table Joins 309
Expressions and Operators 310
Transact-SQL Functions 313
Arithmetic Functions 314
String Functions 315
Date and Time Functions 317
Working with Groups of Values 318
The `COUNT` Function 319
Grouping Records Using `GROUP BY` 319
Filtering Groups Using `HAVING` 321
The `SUM`, `AVG`, `MIN`, and `MAX` Functions 322
Updating Existing Data 322
The `INSERT` Statement 323
The `UPDATE` Statement 324
The `DELETE` Statement 325
Stored Procedures 326
Summary 330

9. ADO.NET 331
Introducing ADO.NET 332
Importing the `SqlClient` Namespace 333
Defining the Database Connection 334
Preparing the Command 336

Executing the Command 337
Setting up Database Authentication 339
Reading the Data 342
Using Parameters with Queries 344
Bulletproofing Data Access Code 351
Using the Repeater Control 354
Creating the Dorknozzle Employee Directory 360
More Data Binding 365
Inserting Records 371
Updating Records 378
Deleting Records 394
Using Stored Procedures 397
Summary 399

10. Displaying Content Using Data Lists 401
`DataList` Basics 402
Handling `DataList` Events 406
Editing `DataList` Items and Using Templates 413
`DataList` and Visual Web Developer 422
Styling the DataList 424
Summary 426

11. Managing Content Using Grid View and Details View 427
Using the `GridView` Control 428
Customizing the `GridView` Columns 435
Styling the GridView with Templates, Skins, and CSS 436
Selecting Grid Records 440
Using the `DetailsView` Control 445
Styling the DetailsView 450
`GridView` and `DetailsView` Events 452
Entering Edit Mode 456
Using Templates 459
Updating `DetailsView` Records 463
Summary 468

12. Advanced Data Access 469
Using Data Source Controls 470
Binding the `GridView` to a SqlDataSource 472
Binding the `DetailsView` to a SqlDataSource 479
Displaying Lists in DetailsView 489
More on SqlDataSource 492
Working with Data Sets and Data Tables 494
What is a Data Set Made From? 497

Binding `DataSets` to Controls 498
Implementing Paging 504
Storing Data Sets in View State 506
Implementing Sorting 509
Filtering Data 520
Updating a Database from a Modified DataSet 521
Summary 526

13. Security and User Authentication 527
Basic Security Guidelines 528
Securing ASP.NET 2.0 Applications 530
Working with Forms Authentication 532
ASP.NET 2.0 Memberships and Roles 544
Creating the Membership Data Structures 544
Using your Database to Store Membership Data 547
Using the ASP.NET Web Site Configuration Tool 552
Creating Users and Roles 554
Changing Password Strength Requirements 556
Securing your Web Application 559
Using the ASP.NET Login Controls 561
Summary 569

14. Working with Files and Email 571
Writing and Reading Text Files 572
Setting Up Security 573
Writing Content to a Text File 576
Reading Content from a Text File 580
Accessing Directories and Directory Information 583
Working with Directory and File Paths 586
Uploading Files 590
Sending Email with ASP.NET 593
Configuring the SMTP Server 595
Sending a Test Email 597
Creating the Company Newsletter Page 601
Summary 610

A. Web Control Reference 611
The `WebControl` Class 611
Properties 611
Methods 612
Standard Web Controls 613
`AdRotator` 613
`BulletedList` 613

Button 614
Calendar 615
CheckBox 617
CheckBoxList 617
DropDownList 619
FileUpload 619
HiddenField 620
HyperLink 620
Image 621
ImageButton 621
ImageMap 622
Label 622
LinkButton 623
ListBox 623
Literal 624
MultiView 624
Panel 625
PlaceHolder 625
RadioButton 625
RadioButtonList 626
TextBox 627
Xml 628
Validation Controls 628
CompareValidator 628
CustomValidator 629
RangeValidator 630
RegularExpressionValidator 631
RequiredFieldValidator 632
ValidationSummary 633
Navigation Web Controls 634
SiteMapPath 634
Menu 635
TreeView 640
HTML Server Controls 643
HtmlAnchor Control 644
HtmlButton Control 644
HtmlForm Control 645
HtmlGeneric Control 646
HtmlImage Control 647
HtmlInputButton Control 647
HtmlInputCheckBox Control 648
HtmlInputFile Control 649

`HtmlInputHidden` Control 650
`HtmlInputImage` Control 651
`HtmlInputRadioButton` Control 652
`HtmlInputText` Control 653
`HtmlSelect` Control 653
`HtmlTable` Control 655
`HtmlTableCell` Control 656
`HtmlTableRow` Control 657
`HtmlTextArea` Control 658
Index 659

Preface

Web development is very exciting. There's nothing like the feeling you have after you place your first dynamic web site online, and see your little toy in action while other people are actually using it!

Web development with ASP.NET is particularly exciting. If you've never created a dynamic web site before, I'm sure you'll fall in love with this area of web development. If you've worked with other server-side technologies, I expect you'll be a little shocked by the differences.

ASP.NET really is a unique technology, and it provides new and extremely efficient ways to create web applications using the programming language with which you feel most comfortable. Though it can take some time to learn, ASP.NET is simple to use. Whether you want to create simple web forms, or feature-rich shopping carts, or even complex enterprise applications, ASP.NET can help you do it. All the tools you'll need to get up and running are immediately available and easy to install, and require very little initial configuration.

This book will be your gentle introduction to the wonderful world of ASP.NET, teaching you the foundations step by step. First, you'll learn the theory; then, you'll put it in practice as we work through practical exercises together. To demonstrate some of the more complex functionality, and to put the theory into a cohesive, realistic context, we'll develop a project through the course of this book. The project—an intranet site for a company named Dorknozzle—will allow us to see the many components of .NET in action, and to understand through practice exactly how .NET works in the real world.

We hope you'll find reading this book an enjoyable experience that will significantly help you with your future web development projects!

Who Should Read this Book?

This book is aimed at beginner, intermediate, and advanced web designers looking to make the leap into server-side programming with ASP.NET. We expect that you'll already feel comfortable with HTML and a little CSS, as very little explanation of these topics is provided here.

By the end of this book, you should be able to successfully download and install ASP.NET and the .NET Framework, configure and start your web server, create and work with basic ASP.NET pages, install and run SQL Server 2005, create

database tables, and work with advanced, dynamic ASP.NET pages that query, insert, update, and delete information within a database.

All examples provided in the book are written in both Visual Basic and C#, the two most popular languages for creating ASP.NET web sites. The examples start at beginners' level and proceed to more advanced levels. As such, no prior knowledge of either language is required in order to read, understand, learn from, and apply the knowledge provided in this book. Experience with other programming or scripting languages (such as JavaScript) will certainly grease the wheels, though, and should enable you to grasp fundamental programming concepts more quickly.

What's in this Book?

This book comprises the following chapters. Read them from beginning to end to gain a complete understanding of the subject, or skip around if you feel you need a refresher on a particular topic.

Chapter 1: Introducing ASP.NET

Before you can start building your database-driven web presence, you must ensure that you have the right tools for the job. In this first chapter, you'll learn how to find, download, and configure the .NET Framework. You'll learn where the web server is located, and how to install and configure it. Next, we'll walk through the installation of the Microsoft database solution: SQL Server 2005. Finally, we'll create a simple ASP.NET page to make sure that everything's running and properly configured.

Chapter 2: ASP.NET Basics

In this chapter, you'll create your first useful ASP.NET page. We'll explore all of the components that make up a typical ASP.NET page, including directives, controls, and code. Then, we'll walk through the process of deployment, focusing specifically on allowing the user to view the processing of a simple ASP.NET page through a web browser.

Chapter 3: VB and C# Programming Basics

In this chapter, we'll look at two of the programming languages that are used to create ASP.NET pages: VB and C#. You'll learn about the syntax of the two languages as we explore the concepts of variables, data types, conditionals, loops, arrays, functions, and more. Finally, we'll see how the two languages accommodate Object Oriented Programming principles by allowing you to work with classes, methods, properties, inheritance, and so on.

Chapter 4: Constructing ASP.NET Web Forms

Web forms are the ASP.NET equivalent of web pages but, as we'll see, the process of building ASP.NET web forms is a lot like composing a castle with Lego bricks! ASP.NET is bundled with hundreds of controls—including HTML controls, web controls, and so on—that are designed for easy deployment within your applications. This chapter will introduce you to these building blocks, and show how to lock them together. You'll also learn about master pages, which are a very exciting new feature of ASP.NET 2.0.

Chapter 5: Building Web Applications

A web application is basically a group of web forms, controls, and other elements that work together to achieve complex functionality. So it's no surprise that when we build web applications, we must consider more aspects than when we build individual web forms. This chapter touches on those aspects, beginning with a hands-on tour of the free IDE from Microsoft, called Visual Web Developer 2005 Express Edition. Next, we configure your web application, learn how to use the application state, user sessions, and cookies, explore the process for debugging errors in your project, and more.

Chapter 6: Using the Validation Controls

This chapter introduces validation controls. With validation controls, Microsoft basically eliminated the headache of fumbling through, and configuring, tired, reused client-side validation scripts. First, we'll learn how to implement user input validation on both the client and server sides of your application using Microsoft's ready-made validation controls. Then, we'll learn how to perform more advanced validation using regular expressions and custom validators.

Chapter 7: Database Design and Development

Undoubtedly one of the most important chapters in the book, Chapter 7 will prepare you to work with databases in ASP.NET. We'll cover the essentials you'll need to know in order to create a database using SQL Server Express Edition. Also in this chapter, we'll begin to build the database for the Dorknozzle intranet project.

Chapter 8: Speaking SQL

This chapter will teach you to speak the language of the database: Structured Query Language, or SQL. After a gentle introduction to the basic concepts of SQL, which will teach you how to write `SELECT`, `INSERT`, `UPDATE`, and `DELETE` queries, we'll move on to more advanced topics such as expressions, conditions, and joins. Finally, we'll take a look at how we can reuse queries quickly and easily by writing stored procedures.

Chapter 9: ADO.NET

The next logical step in building database-driven web applications is to roll up our sleeves and dirty our hands with a little ADO.NET—the technology that facilitates communication between your web application and the database server. This chapter explores the essentials of the technology, and will have you reading database data directly from your web applications in just a few short steps. We'll then help you begin the transition from working with static applications to those that are database-driven.

Chapter 10: Displaying Content Using Data Lists

Taking ADO.NET further, this chapter shows you how to utilize the `DataList` control provided within the .NET Framework. `DataLists` play a crucial role in simplifying the presentation of information with ASP.NET. In learning how to present database data within your applications in a cleaner and more legible format, you'll gain an understanding of the concepts of data binding at a high level.

Chapter 11: Managing Content Using GridView and DetailsView

This chapter explores two of the most powerful data presentation controls of ASP.NET: `GridView` and `DetailsView`. `GridView` supersedes ASP.NET 1.x's `DataGrid`, and is a very powerful control that automates almost all tasks that involve displaying grids of data. `DetailsView` completes the picture by offering us the functionality needed to display the details of a single grid item.

Chapter 12: Advanced Data Access

This chapter explores a few of the more advanced details involved in data access, retrieval, and manipulation. We'll start by looking at direct data access using ADO.NET's data source controls. We'll then compare this approach with that of using data sets to access data in a disconnected fashion. In this section, you'll also learn to implement features such as paging, filtering, and sorting using custom code.

Chapter 13: Security and User Authentication

This chapter will show you how to secure your web applications with ASP.NET. We'll discuss the various security models available, including IIS, Forms, Windows, and Passport, and explore the roles that the Web.config and XML files can play. This chapter will also introduce you to the new ASP.NET 2.0 membership model, and the new ASP.NET 2.0 login controls.

Chapter 14: Working with Files and Email

In this chapter, we'll look at the task of accessing your server's file system, including drives, files, and the network. Next, the chapter will show you how to work with file streams to create text files, write to text files, and read from text files stored on your web server. Finally, you'll get first-hand experience in sending emails using ASP.NET.

Appendix

Included in this book is a handy web control reference, which lists the most common properties and methods of the most frequently used controls in ASP.NET.

The Book's Web Site

Located at http://www.sitepoint.com/books/aspnet2/, the web site that supports this book will give you access to the following facilities.

The Code Archive

As you progress through this book, you'll note a number of references to the code archive. This is a downloadable ZIP archive that contains complete code for all the examples presented in the book. You can get it from the book's web site.[1]

The archive contains one folder for each chapter of this book. Each folder contains CS and VB subfolders, which contain the C# and VB versions of all the examples for that chapter, respectively. In later chapters, these files are further divided into two more subfolders: Lessons for standalone examples presented for a single chapter, and Project for files associated with the Dorknozzle Intranet Application, the project that we'll work on throughout the book.

Updates and Errata

No book is perfect, and we expect that watchful readers will be able to spot at least one or two mistakes before the end of this one. The Errata page on the book's web site will always have the latest information about known typographical and code errors, and necessary updates for new releases of ASP.NET and the various web standards that apply.

[1] http://www.sitepoint.com/books/aspnet2/code.php

The SitePoint Forums

If you'd like to communicate with us or anyone else on the SitePoint publishing team about this book, you should join SitePoint's online community.[2] The .NET forum, in particular, can offer an abundance of information above and beyond the solutions in this book.[3]

In fact, you should join that community even if you don't want to talk to us, because a lot of fun and experienced web designers and developers hang out there. It's a good way to learn new stuff, get questions answered in a hurry, and just have fun.

The SitePoint Newsletters

In addition to books like this one, SitePoint publishes free email newsletters including *The SitePoint Tribune* and *The SitePoint Tech Times*. In them, you'll read about the latest news, product releases, trends, tips, and techniques for all aspects of web development. If nothing else, you'll get useful ASP.NET articles and tips, but if you're interested in learning other technologies, you'll find them especially valuable. Sign up to one or more SitePoint newsletters at http://www.sitepoint.com/newsletter/.

Your Feedback

If you can't find your answer through the forums, or if you wish to contact us for any other reason, the best place to write is `books@sitepoint.com`. We have a well-manned email support system set up to track your inquiries, and if our support staff members are unable to answer your question, they will send it straight to us. Suggestions for improvements, as well as notices of any mistakes you may find, are especially welcome.

Acknowledgements

First and foremost, I'd like to thank the SitePoint team for doing such a great job in making this book possible, for being understanding as deadlines inevitably slipped past, and for the team's personal touch, which made it a pleasure to work on this project.

[2] http://www.sitepoint.com/forums/
[3] http://www.sitepoint.com/forums/forumdisplay.php?f=141

Particular thanks go to Simon Mackie, whose valuable insight and close cooperation throughout the process has tied up many loose ends and helped make this book both readable and accessible. Thanks again Simon for allowing me to write this book—I appreciate the patience and dedication you've shown.

Finally, returning home, I'd like to thank my wife Jessica, whose patience, love, and understanding throughout continue to amaze me.

—Zak Ruvalcaba

I'd like to thank Simon Mackie, the Managing Editor at SitePoint, for being extremely supportive during the process of writing this book. Warm thanks and gratitude go to my parents, my girlfriend, and my close friends for constantly being there for me.

—Cristian Darie

1 Introducing ASP.NET and the .NET Platform

ASP.NET is one of the most exciting web development technologies on offer today. When Microsoft released the first version a few years ago, many web developers thought all their dreams had come true. Here was a powerful platform with lots of built-in functionality, astonishing performance levels, and one of the best IDEs (Integrated Development Environments) around: Visual Studio. What more could anyone want? Indeed, ASP.NET showed the way for the faster, easier, and more disciplined development of dynamic web sites, and the results were impressive.

Time has passed, and ASP.NET has grown. ASP.NET 2.0 comes with extraordinary new features as well as an expanded and more powerful underlying framework. Not only that, but the basic versions of all development tools, including Visual Web Developer 2005 Express Edition and SQL Server 2005 Express Edition, are free!

This book shows you how to use all these technologies together in order to produce fantastic results. We'll take you step by step through each task, showing you how to get the most out of each technology and tool. Developers who have already worked with earlier versions of ASP.NET will find that the latest version has changed so much that entire chapters of this book are devoted to ASP.NET 2.0-specific features.

Let's begin!

What is ASP.NET?

For years, the Active Server Pages (ASP) technology was arguably the leading choice for web developers building dynamic web sites on Windows web servers, as it offered flexible yet powerful scripting capabilities. Early in 2002, Microsoft released a new technology for Internet development called ASP.NET. ASP.NET represents a leap forward from ASP both in its sophistication and the productivity gains it achieves for developers. It continues to offer flexibility in terms of language support, but rather than a range of simple scripting languages, several fully-fledged programming languages are now at the fingertips of ASP.NET developers. Development in ASP.NET requires not only an understanding of HTML and web design, but also a firm grasp of the concepts of object oriented programming and development.

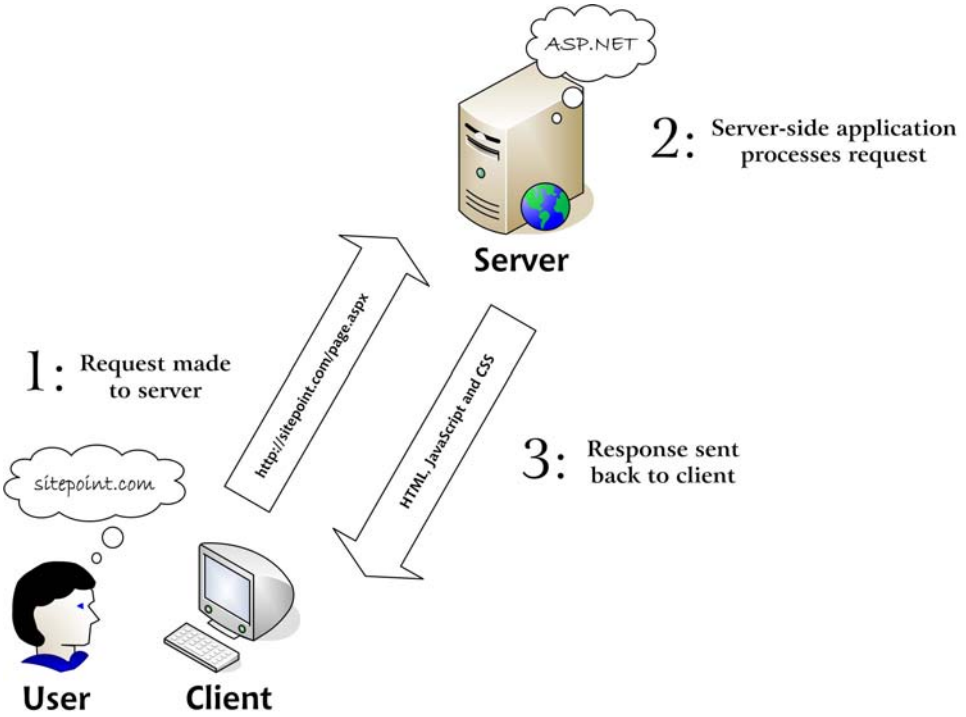
In the next few sections, we'll introduce you to the basics of ASP.NET. We'll walk through the process of installing it on your web server, and step through a simple example that demonstrates how ASP.NET pages are constructed. But first, let's define what ASP.NET actually is.

ASP.NET is a server-side technology for developing web applications based on the Microsoft .NET Framework. Let's break that jargon-filled sentence down.

ASP.NET is a server-side technology; that is, it runs on the web server. Most web designers start their careers learning client-side technologies like HTML, JavaScript, and Cascading Style Sheets (CSS). When a web browser requests a web page created with only client-side technologies, the web server simply grabs the files that the browser (or client) requests and sends them down the line. The client is entirely responsible for reading the markup in those files and interpreting that markup to display the page on the screen.

Server-side technologies, like ASP.NET, are different. Instead of being interpreted by the client, server-side code (for example, the code in an ASP.NET page) is interpreted by the web server. In the case of ASP.NET, the code in the page is read by the server and used to generate HTML, JavaScript, and CSS that is then sent to the browser. Since the processing of the ASP.NET code occurs on the server, it's called a server-side technology. As Figure 1.1 shows, the client only sees the HTML, JavaScript, and CSS. The server is entirely responsible for processing the server-side code.

Figure 1.1. A user interacting with a web application



Note the three roles involved in such a transaction:

user Never forget that there's a real person at the end (or beginning) of the line.

web client This is the software program that the person uses to interact to the web application. The client is usually a web browser, such as Internet Explorer or Firefox.

web server This is the software program located on the server. It processes requests made by the web client.

ASP.NET is a technology for developing web applications. A web application is just a fancy name for a dynamic web site. Web applications usually (but not always) store information in a database, and allow visitors to the site to access and change that information. Many different programming technologies and supported languages have been developed to create web applications; PHP, JSP, Ruby on Rails,

CGI, and ColdFusion are just a few of the more popular ones. However, rather than tying you to a specific technology and language, ASP.NET lets you write web applications using a variety of familiar programming languages.

ASP.NET uses the Microsoft .NET Framework. The .NET Framework collects all the technologies needed for building Windows desktop applications, web applications, web services, and so on, into a single package, and makes them available to more than 40 programming languages.

Even with all the jargon explained, you're probably still wondering what makes ASP.NET so good. The truth is that there are many server-side technologies around, each of which has its own strengths and weaknesses. Yet ASP.NET has a few features that really are unique:

- ASP.NET lets you use your favorite programming language, or at least one that's close to it. The .NET Framework currently supports over 40 languages, and many of these may be used to build ASP.NET web sites. The most popular choices are C# (pronounced "C sharp") and Visual Basic (or VB), which are the ones we'll cover in this book.

- ASP.NET pages are *compiled*, not interpreted. In ASP.NET's predecessor, ASP, pages were interpreted: every time a user requested a page, the server would read the page's code into memory, figure out how to execute the code (that is, interpret the code), and execute it. In ASP.NET, the server need only figure out how to execute the code once. The code is compiled into efficient binary files, which can be run very quickly, again and again, without the overhead involved in re-reading the page each time. This represents a big jump in performance from the old days of ASP.

- ASP.NET has full access to the functionality of the .NET Framework. Support for XML, web services, database interaction, email, regular expressions, and many other technologies are built right into .NET, which saves you from having to reinvent the wheel.

- ASP.NET allows you to separate the server-side code in your pages from the HTML layout. When you're working with a team composed of programmers and design specialists, this separation is a great help, as it lets programmers modify the server-side code without stepping on the designers' carefully crafted HTML—and vice versa.

- ASP.NET makes it easy to reuse common User Interface elements in many web forms, as it allows us to save those components as independent web user controls. During the course of this book, you'll learn how to add powerful

features to your web site, and to reuse them in many places with a minimum of effort.

- You can get excellent tools that assist in developing ASP.NET web applications. Visual Web Developer 2005 is a free, powerful visual editor that includes features such as code autocompletion, code formatting, database integration functionality, a visual HTML editor, debugging, and more. In the course of this book, you'll learn how to use this tool to build the examples we discuss.

- The .NET Framework was first available only to the Windows platform, but thanks to projects such as Mono,[1] it's since been ported to other operating systems.

Still with me? Great! It's time to gather our tools and start building!

Installing the Required Software

If you're going to learn ASP.NET, you first need to make sure you have all the necessary software components installed and working on your system. Let's take care of this before we move on.

Internet Information Services (IIS) or Cassini
: IIS is the web server of choice for running ASP.NET web applications. You'll need your copy of the Windows CD to install and configure it. Unfortunately, some versions of Windows (such as Windows XP Home Edition) don't support IIS. If you're one of those users, there's **Cassini**. Cassini is a small web server designed for hobbyists who are looking to build ASP.NET web sites. It isn't as robust, powerful, or user-friendly as IIS, but it will be sufficient for our purposes. When we come to use Visual Web Developer in Chapter 5, we'll be making use of that product's built-in development web server, so not having access to IIS on your system won't be a problem.

a modern web browser
: Throughout this book, we'll be using Internet Explorer 6, but you can use other browsers during development if you wish. Any modern browser will do.

.NET Framework 2.0
: As we've already discussed, the .NET Framework drives ASP.NET. When you install the .NET Framework, you'll automatically install the files necessary

[1] http://www.mono-project.com/

to run ASP.NET. You're likely to have the .NET Framework already, as it installs automatically through the Windows Update service.

.NET Framework Software Development Kit (SDK)
The .NET Framework 2.0 Software Development Kit (SDK) is a free download that contains the necessary Web Application development tools, a debugger for error correcting, and a suite of samples and documentation.

We're also going to need a database. In this book, we'll use the following:

Microsoft SQL Server 2005 Express Edition
This is the free, but still fully functional, version of SQL Server 2005. If you worked with previous versions of these technologies, you should know that SQL Server 2005 Express is a replacement for the previous Microsoft SQL Data Engine (MSDE). You can read more on the differences between various SQL Server 2005 editions at the Microsoft site.[2]

SQL Server Management Studio Express
Because the Express Edition of SQL Server doesn't ship with any visual management tools, you can use this free tool, also developed by Microsoft, to access your SQL Server 2005 databases.

Installing the Web Server

Installing Internet Information Services (IIS)

IIS comes with most versions of server-capable Windows operating systems—including Windows 2000 Professional, Server, and Advanced Server; Windows XP Professional; Windows XP Media Center Edition; and Windows Server 2003—but it's not installed automatically in all versions, which is why it may not be present on your computer. IIS isn't available for Home editions of these operating systems, such as Windows XP Home Edition. If you run this, you'll need to rely on Cassini, which we discuss below.

To see whether you have IIS installed and running, simply locate your Administrative Tools folder (sometimes it's a menu option; sometimes it's a folder in the Control Panel[3]) and check whether or not it contains a shortcut to Internet Information Services. If the shortcut isn't visible, then it's not installed. To install IIS, simply follow these steps:

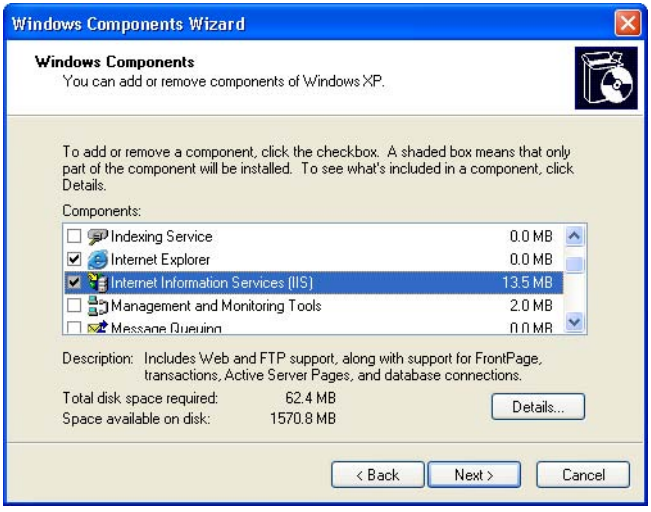
[2] http://www.microsoft.com/sql/2005/productinfo/sql2005features.asp
[3] To see this folder, you'll need to view the Control Panel in "classic view."

1. In the Control Panel, select Add or Remove Programs.

2. Choose Add/Remove Windows Components. The list of components will become visible within a few seconds.

3. In the list of components, check Internet Information Services (IIS), as shown in Figure 1.2. The default installation options are enough for ASP.NET development, but you may want to click Details... to view the extra options you could add.

Figure 1.2. Installing IIS



4. Click Next. Windows may prompt you to insert the Windows CD.

Add Administrative Tools to the Start Menu

Here's how to add Administrative Tools to the Windows XP Start menu:

1. Right-click on the Start button and select Properties to bring up the Taskbar and Start Menu Properties dialog.

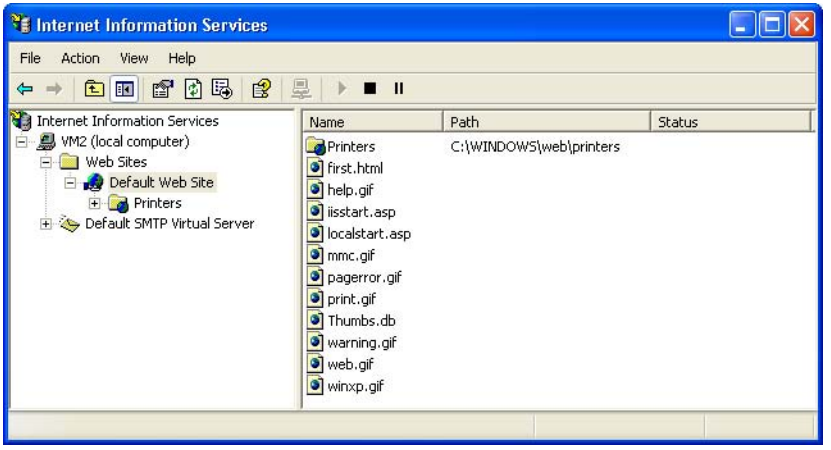
2. Click the Customize... button to bring up the Customize Start Menu dialog.

3. If you're using the classic Start menu, check Display Administrative Tools, then click OK.

4. If you're using the Windows XP-style Start menu, click the Advanced tab, scroll through the Start menu items list until you get to System Administrative Tools, and select from Display on the All Programs menu or Display in the All Programs menu and the Start menu.

Once IIS is installed, close the Add or Remove Programs dialog. To check that IIS has installed correctly, see if you can find the Internet Information Services short cut in Administrative Tools. If you can, IIS is installed. Open the link to make first contact with the IIS management console, which is shown in Figure 1.3. In the left pane, you'll initially see the name of your computer, whose nodes you can expand.

Figure 1.3. The IIS administration tool



You can close this tool for now; you'll meet it again later.

You are now ready to host web applications. Although we won't cover the configuration of IIS for external use, we will show you how to configure IIS to support local development of ASP.NET applications in order that they may be uploaded to your external web hosting provider later.

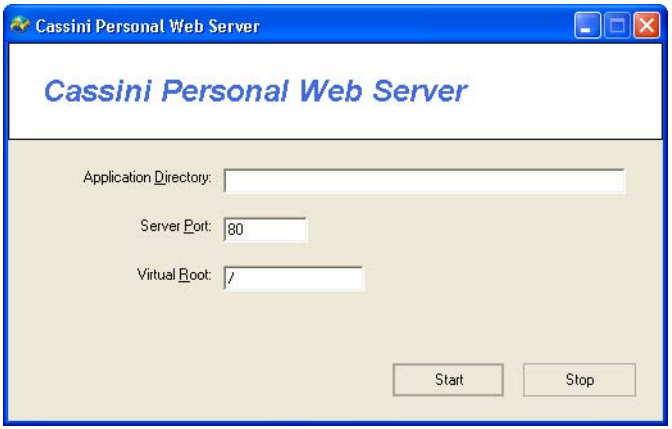
Installing Cassini

If you're unable to install IIS, you'll need to download and install Cassini:

1. Go to the Cassini download page.[4]
2. Download the Cassini installer executable (`cassini.exe`).
3. Run `cassini.exe` and follow the steps presented by the installer, accepting the default options.

If the process went to plan, everything you need to run Cassini can be found in the folder `C:\Cassini`. Double-click `CassiniWebServer.exe` in that folder to start its management console, which is shown in Figure 1.4.

Figure 1.4. The Cassini management console



We'll need to do some more work to get Cassini up and running properly, but we'll need to install the .NET Framework and the Software Development Kit first.

Installing the .NET Framework and the SDK

To begin creating ASP.NET applications, you'll need to install the .NET Framework and the Software Development Kit (SDK). The .NET Framework includes

[4] http://www.asp.net/Projects/Cassini/Download/

the files necessary to run and view ASP.NET pages, while the SDK includes samples, documentation, and a variety of free tools.

Installing the .NET Framework

The best method of acquiring the .NET Framework is to download and install it directly from the Web. Note that it is also delivered through Windows Update, so you may already have it installed on your system. To check, open the folder `C:\WINDOWS\Microsoft.NET\Framework` (if your copy of Windows is installed somewhere other than `C:\WINDOWS`, change this path accordingly). If this folder doesn't exist, you definitely don't have the .NET Framework installed. If it does exist, you should find inside it at least one folder with a name like `v1.1.4322`. Each of these kinds of folders holds a different version of the .NET Framework you have installed. If at least one of these folders' names doesn't start with `v2` or higher, you'll need to install the latest version of the .NET Framework.

To install the latest version of the .NET Framework, simply follow the steps outlined below:

1. Go to the ASP.NET support site[5] and click the Download the .NET Framework link.
2. Under the .NET Framework Version 2.0 Redistributable Package heading, click the appropriate download link for your hardware. Remember, we'll install the redistributable package first, then the SDK. The link will advance you to a download page.
3. Choose the language and version of the installation you want, and click Download.
4. Save the file to a local directory. After the download is complete, double-click the executable to begin the installation.
5. Follow the steps presented by the wizard until installation completes.

Installing the SDK

Now that you've installed the redistributable package, you need to install the Software Development Kit (SDK):

[5] http://www.asp.net/

1. Go back to the ASP.NET support site and follow the Download the .NET Framework link again.

2. This time, click the appropriate download link under the .NET Framework Version 2.0 Software Development Kit heading. The link will advance you to a download page.

3. Choose the language version of the installation you want to use and click **Download**, as you did to download the redistributable package.

4. When prompted to do so, save the file to a local directory.

5. After the download is complete, double-click the executable to begin the installation. Before you do so, I strongly recommend that you close all other programs to ensure the install proceeds smoothly.

6. Follow the steps outlined by the .NET Setup Wizard until installation completes. When asked for setup options, it's safe to use the default values.

The SDK will take slightly longer to install than the framework.

A Big Download!

The .NET Framework SDK weighs in at about 350MB, so it will probably take a while to download.

Configuring the Web Server

Configuring IIS

After installing the .NET Framework and the SDK manually, you will need to configure IIS to make it aware of ASP.NET. To do this, you need to follow a few simple steps:

1. Open the command prompt by selecting Start > All Programs > Microsoft .NET Frameworks SDK v2.0 > SDK Command Prompt.

2. Type the following command to install ASP.NET:

```
C:\Program Files\…\SDK\v2.0>aspnet_regiis.exe -i
Start installing ASP.NET (2.0.50727).
.........
Finished installing ASP.NET (2.0.50727).
```

3. Once ASP.NET is installed, close the command prompt and check again to confirm that ASP.NET installed correctly.

Running `aspnet_regiis.exe`

Depending on the circumstances, ASP.NET may already have been installed for you, but running `aspnet_regiis.exe` can't hurt. Also, remember that you need to run this utility again in case you reinstall IIS.

Configuring Cassini

If you've installed Cassini, you'll need to get under the hood of the .NET Framework to coerce Cassini into working as it should.

1. Open the command prompt by selecting Start > All Programs > Microsoft .NET Frameworks SDK v2.0 > SDK Command Prompt.

2. Enter the following command at the prompt:

```
C:\Program Files\…\SDK\v2.0>gacutil /i C:\Cassini\Cassini.dll
Microsoft (R) .NET Global Assembly Cache Utility. Version 2.0…
Copyright (c) Microsoft Corporation.  All rights reserved.

Assembly successfully added to the cache
```

Cassini is now ready to go.

Where do I Put my Files?

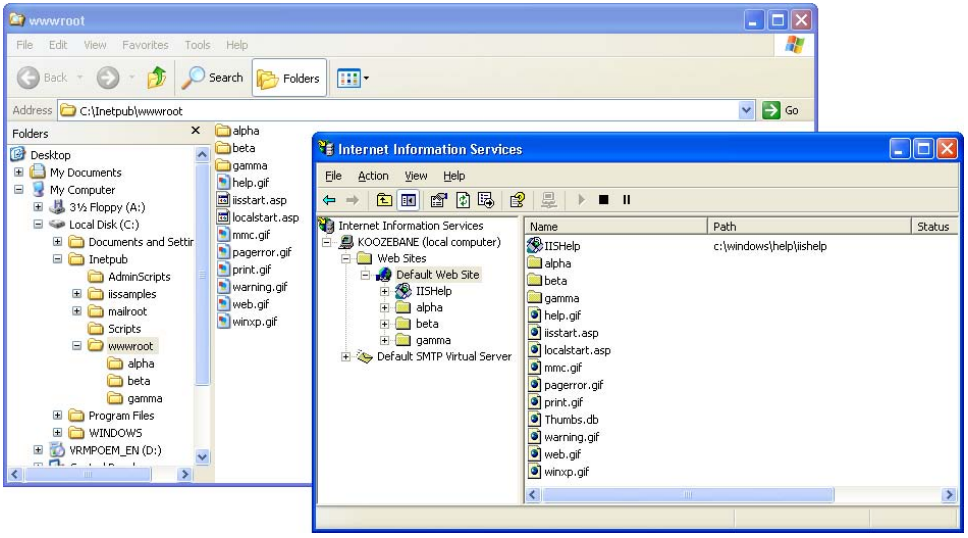
IIS Recommended

From here on in, the instructions we provide will be centered around IIS, as Cassini isn't suitable for production environments. Many of the concepts we'll discuss do not apply to Cassini, as it's much simpler and lacks many of IIS's features. Where needed, Cassini instructions will be given, but IIS will receive the bulk of the discussion.

Now that you have ASP.NET up and running, let's find out where the files for your web applications are kept on the computer. You can readily set IIS to look for web applications in any folder, including the `My Documents` folder, or even a network share. By default, IIS maps the `C:\Inetpub\wwwroot` folder of your disk to your web site's root directory, which is generally considered a good repository for storing and managing your web applications.

If you open this `wwwroot` folder in Windows Explorer, and compare its contents with the files that appear in the Default Web Site in the IIS administration tool, as shown in Figure 1.5, you'll notice that the files and folders are the same (some extra items will be listed in IIS; we'll look at these shortly). You need to use the IIS administration tool to set up the behavior of these files and folders under IIS. We'll see more on this soon.

Figure 1.5. Folders inside `wwwroot` also appear inside IIS



Using localhost

By putting your files within `C:\Inetpub\wwwroot`, you give your web server access to them. If you've been developing web pages for a long time, habit may drive you to open files directly in your browser by double-clicking on the HTML files. However, because ASP.NET is a server-side language, your web server needs to have a crack at the file before it's sent to your browser for display. If the server doesn't get this opportunity, the ASP.NET code won't be converted into HTML that your browser can understand. For this reason, ASP.NET files can't be opened directly from the disk using Windows Explorer.

Your local web server can be accessed through a special web address that indicates the current computer: http://localhost/. If you try this now, IIS will open up a default help page (although this behavior will vary depending on the settings of

your Windows installation; for example, if you get an error instead of the default help page, don't worry).

What you need to keep in mind, though, is that the address you'll use to access local web applications will always start with http://localhost/, and that, by default, this root address points to the folder on your disk.

To see this in practice, create a new file named `index.htm` inside `C:\Inetpub\wwwroot`, with the following contents[6]:

File: **index.htm**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Simple HTML Page</title>
  </head>
  <body>
    <P>This is a simple HTML page.
  </body>
</html>
```

Now, load this page through `http://localhost/index.htm`, as shown in Figure 1.6.

Figure 1.6. Testing IIS

Experiencing an Error?

If the page doesn't load as illustrated in Figure 1.6, your IIS installation has problems. You might want to double-check that you correctly followed the steps for installing it, and re-check the IIS configuration procedure.

[6] All of the code and images used in this book are available for download from sitepoint.com. See the Preface for more information.

This localhost name is equivalent to the so-called loopback IP address, 127.0.0.1, so you can get the same results by entering **`http://127.0.0.1/index.htm`** into your browser. If you know them, you can also use the name or IP address of your machine to the same end.

Note that if you do try any of these equivalents, a dialog will appear before the page is opened, to ask you for your network credentials. This occurs because you're no longer using your local authentication, which is implicit with localhost.

Stopping and Starting IIS

Now that we have IIS up and running, and ASP.NET installed, let's look at how you can start, stop, and restart IIS if the need arises. For the most part, you'll always want to have IIS running; however, if you want to shut it down temporarily for any reason (such as security concerns), you can. Also, some external programs may stop IIS upon launch because of potential security vulnerabilities, so you'll need to start it again yourself. If you want to stop IIS when it's not being used, simply open the Internet Information Services management console, right-click on Default Web Site and select Stop. Alternatively, after selecting Default Web Site, you can use the Stop, Pause, and Play icons from the toolbar.

Virtual Directories

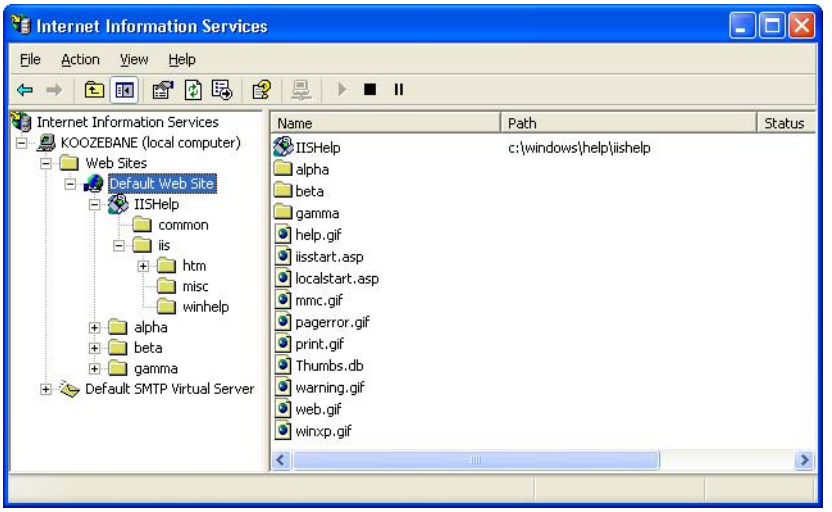
As we saw in the section called "Where do I Put my Files?", physical sub-folders of `C:\Inetpub\wwwroot` also become subdirectories of the web site. For instance, imagine your company has a web server that serves documents from `C:\Inetpub\wwwroot`. Your users can access these documents through `http://www.example.com/`. If you create a subfolder of `wwwroot`, named `about`, files in that directory can be accessed via `http://www.example.com/about/`.

You could also set up another subdirectory in your web site, but serve files from a different location on the disk. If, for instance, you were developing another web application, you could store the files for it in `C:\dev\OrderSystem`. You could then create within IIS a new virtual directory called, say, `order`, which mapped to this location. This new site would then be accessible through the URL `http://www.example.com/order/`. As this application is in development, you would probably want to set IIS to hide this virtual directory from the public until the project is complete; your existing web site would still be visible.

By default, a virtual directory, called `IISHelp`, is preconfigured in IIS; it maps to `c:\windows\help\iishelp`. You can see in Figure 1.7 that `IISHelp` contains

subdirectories called `common` and `iis`—these are physical folders inside `c:\windows\help\iishelp`.

Figure 1.7. The `IISHelp` virtual directory



Let's create a virtual directory on your server, and test it with a simple page:

1. First, you need to create on your disk a folder to which your virtual directory will be mapped. Create a folder called `WebDocs` in an easily accessible location on your disk, such as `C:\`, and create a folder named `Learning` inside that folder. We'll use this folder, `C:\WebDocs\Learning`, for various exercises in this book.

2. Copy the `index.htm` file you created earlier into your newly created `Learning` folder.

3. In the Internet Information Services management console, right-click Default Web Site and select New > Virtual Directory. The Virtual Directory Creation Wizard will appear. Click Next.

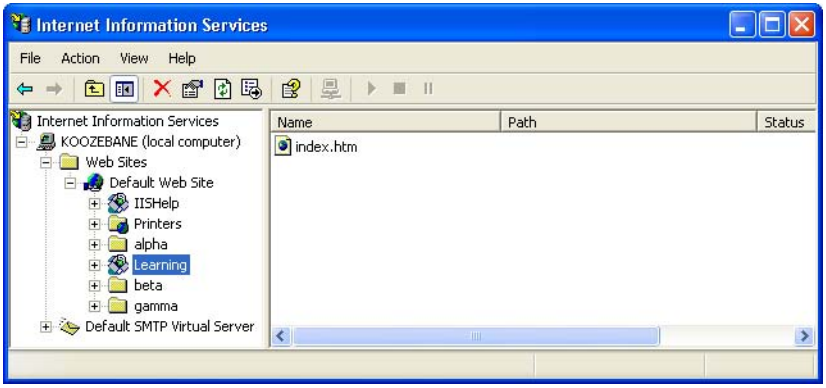
4. You need to choose an alias for your virtual directory: enter **`Learning`**, then click Next.

5. Browse and select the `Learning` folder you created at step 1, or enter its full path (**`C:\WebDocs\Learning`**). Click Next.

6. In the next screen, you can select permissions settings for your directory. Typically, you'll want to leave the default options (Read and Run scripts) checked. Click Next.

7. Click Finish. You'll see your new virtual directory as a child of Default Web Site, as Figure 1.8 illustrates.

Figure 1.8. Creating a new virtual directory



8. Load this link by entering **`http://localhost/Learning/index.htm`** into the address bar of your browser. If everything went well, you should see your little HTML page load, as has the one in Figure 1.9.

Figure 1.9. Testing your new virtual directory

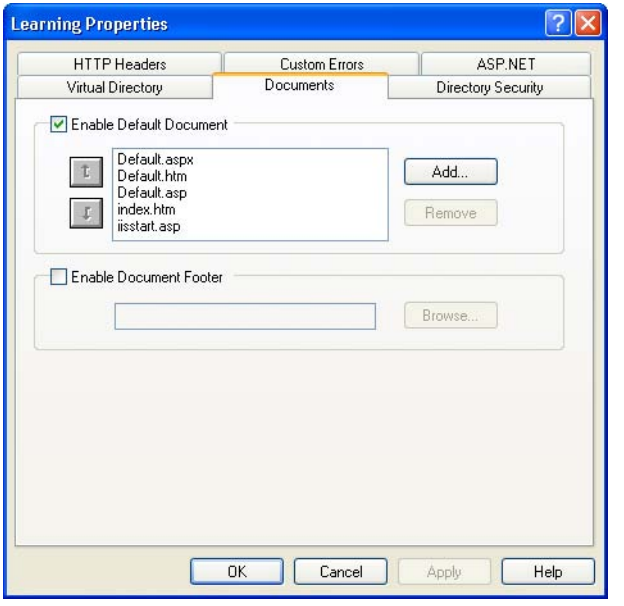
Note that by loading the page through the HTTP protocol, your request goes through IIS. Since `index.htm` is a simple HTML page that doesn't need any server-side processing, you can also load it directly from the disk. However, this

won't be the case with the ASP.NET scripts you'll see through the rest of this book.

Once your new virtual directory has been created, you can see and configure it through the Internet Information Services management console shown in Figure 1.8. You can see the folder's contents in the right-hand panel.

As `index.htm` is one of the default document names, you can access that page just by entering **`http://localhost/Learning/`** into your browser's address bar. To see and edit the default document names for a virtual directory (or any directory, for that matter), you can right-click the directory's name in the IIS management console, click Properties, and select the Documents tab. You'll see the dialog displayed in Figure 1.10.

Figure 1.10. Default document types for the Learning virtual directory



By default, when we request a directory without specifying a filename, IIS looks for a page with the name of one of the default documents, such as `index.htm` or `default.htm`. If there is no index page, IIS assumes we want to see the contents of the requested location. This operation is allowed only if the Directory Browsing

option is selected for the directory in question. You'll find that option in the Directory tab of the Properties window.

Directory Browsing

Enabling directory browsing is not something you'd usually want to do. Allowing visitors to freely see and access all the files and directories that make up your web page is not only a little messy and unprofessional, but also increases the potential for security issues (you don't want any hackers to stick their nose into your code, do you?). So, by default, IIS won't allow directory browsing when a directory is requested: if a default file such as `index.htm` isn't there, ready to be served to the visitor, a message reading "Directory Listing Denied" will be served instead.

To change your virtual directory's options, you have to right-click the virtual directory (`Learning`, in our case) in the IIS console, and choose Properties. The Properties dialog that we've just used lets us configure various useful properties, including:

Virtual Directory — This option allows you to configure directory-level properties, including path information, the virtual directory name, access permissions, etc. Everything that was set up through the wizard is modifiable through this tab.

Documents — This option allows you to configure a default page that displays when the user types in a full URL. For instance, because `default.aspx` is listed as a default page, the user need only enter **`http://www.mysite.com/`**, rather than **`http://www.mysite.com/default.aspx`**, into the browser's address bar. You can easily change and remove these default pages by selecting the appropriate button to the right of the menu.

Directory Security — This option provides you with security configuration settings for the virtual directory.

HTTP Headers — This option gives you the ability to forcefully control page caching on the server, add custom HTTP Headers, Edit Ratings (this helps identify the content your site provides to users), and create MIME types. Don't worry about this for now.

Custom Errors This option allows you to define your own custom error pages. Rather than presenting the standard error messages that appear within Internet Explorer, you can customize error messages with your company's logo and messages of your choice.

ASP.NET This tab allows you to configure the options for the ASP.NET applications stored in that folder.

One thing to note at this point is that we can set properties for the Default Web Site node, and choose to have them "propagate" down to all the virtual directories we've created.

Using Cassini

If you're stuck using a version of Windows that doesn't support IIS, you'll need to make use of Cassini to get your simple ASP.NET web applications up and running. Cassini doesn't support virtual directories, security settings, or any of IIS's other fancy features; it's just a very simple web server that gives you the basics you need to get up and running.

To get started using Cassini:

1. Create a directory called `C:\WebDocs\Learning`, just like the one we created in the section called "Virtual Directories".

2. Copy `index.htm` into this folder. We first saw `index.htm` in the section called "Using localhost".

3. Start Cassini by opening `C:\Cassini` (or, if you chose to install Cassini somewhere else, open that folder), then double-click on the file `CassiniWebServer.exe`.

4. Cassini has just three configuration options:

   Application Directory
   It's here that your application's files are stored. Enter **`C:\WebDocs\Learning`** into this field.

   Server Port
   Web servers almost always operate on port 80, so we won't touch this setting.

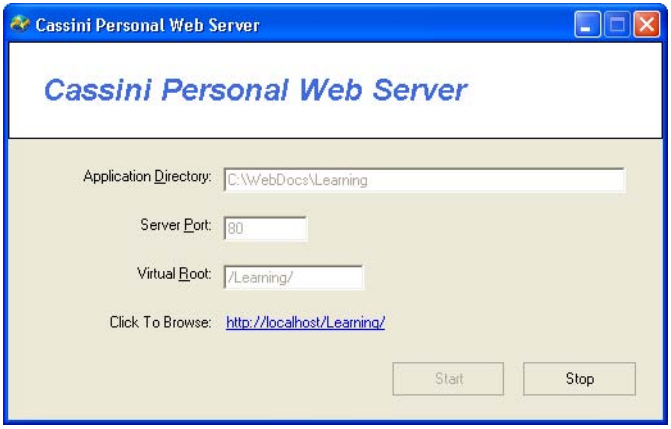
Virtual Root

This is similar to IIS's virtual directories feature, though it's nowhere near as flexible. By default, it's set to `/`, meaning that you can access the file `C:\WebDocs\Learning\index.htm` by entering the address **`http://localhost/index.htm`**. However, to match our IIS virtual directory configuration, we want to make this file's address `http://localhost/Learning/index.htm`. To create this effect, enter **`/Learning/`** into this field.

5. Once you have filled in the Application Directory and Virtual Root fields, click the Start button to start the web server.

6. After clicking Start, a link to the web site that's being served will appear toward the bottom of the window, as shown in Figure 1.11.

Figure 1.11. Cassini serving a web site



When you browse open this site in Cassini, you won't see `index.htm` straight away—you'll be presented with a list of files instead. Cassini only recognizes files named `default.htm` or `default.aspx` as default documents, and it doesn't allow you to configure this feature as IIS does.

Installing SQL Server 2005 Express Edition

After making sure IIS, the .NET Framework, and the SDK are installed correctly, it's time to move forward and install the next piece of software that you'll be using as we work through this book: SQL Server 2005 Express Edition.

SQL Server 2005 is Microsoft's database solution for medium to large companies and enterprises. SQL Server 2005 can be quite expensive, it generally requires its own dedicated database server machine, and, at times, it necessitates that a certified database administrator (DBA) be employed to ensure its maintenance; yet it does offer a robust and scalable solution for larger web applications.

For the examples in this book, we'll use SQL Server 2005 Express Edition, which is free and sufficiently powerful for our needs. Unlike the expensive versions, SQL Server 2005 Express Edition doesn't ship with visual management utilities, but you can use another free tool from Microsoft—SQL Server Management Studio Express, which we'll install next)—for these purposes.

You can install SQL Server 2005 Express Edition as follows:

1. Navigate to http://msdn.microsoft.com/vstudio/express/sql/, and click the Download Now link.

2. In the next page, you can choose between SQL Server 2005 Express Edition, and SQL Server 2005 Express Edition with Advanced Services. The former will be fine for our purposes. Your system should meet the necessary requirements, so go ahead and click Download.

3. Once the download has completed, double-click the downloaded executable file, and follow the steps to install the product. It's safe to use the default options all the way through, though it is a rather long process.

Provided that everything goes well, SQL Server 2005 Express Edition will be up and running at the end of the process. Like IIS, SQL Server runs as a service in the background, accepting connections to databases instead of web pages. The SQL Server is accessible at the address `(local)\SqlExpress`.

Installing SQL Server Management Studio Express

In order to use your SQL Server 2005 install effectively, you'll need some sort of administration tool that will allow you to work with your databases. SQL

Server Management Studio Express is a free tool provided by Microsoft to allow you to manage your installation of SQL Server 2005.

To install SQL Server Management Studio Express, follow these steps:

1. Navigate again to http://msdn.microsoft.com/vstudio/express/sql/, and click the Download Now link.

2. This time, download the SQL Server Management Studio Express edition that corresponds to the SQL Server 2005 version that you installed previously.

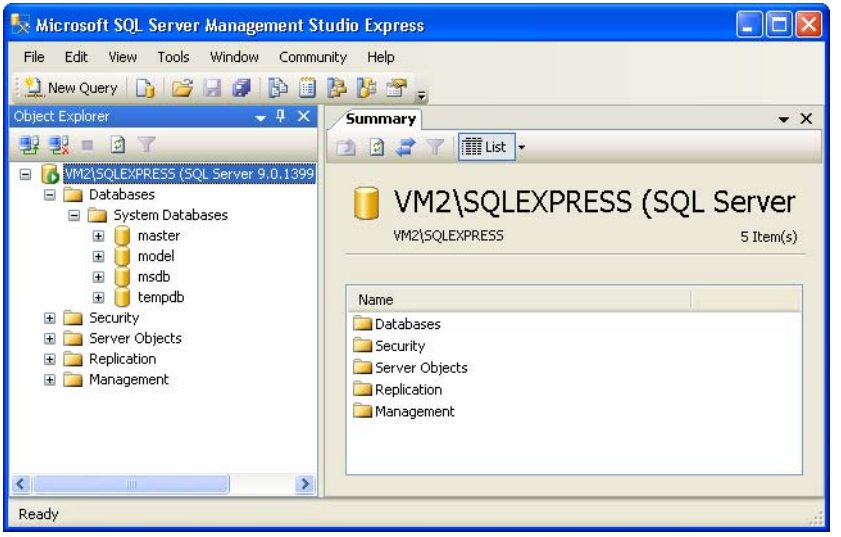
3. After the download completes, execute the file and follow the steps to install the product.

Once it's installed, SQL Server Manager Express can be accessed from Start > All Programs > Microsoft SQL Server 2005 > SQL Server Management Studio Express. When executed, it will first ask for your credentials, as Figure 1.12 illustrates.

Figure 1.12. Connecting to SQL Server

By default, when installed, SQL Server 2005 Express Edition will only accept connections that use Windows Authentication, which means that you'll use your Windows user account to log in to the SQL Server. Because you're the user that installed SQL Server 2005, you'll already have full privileges to the SQL Server. Click Connect to connect to your SQL Server 2005 instance.

Figure 1.13. Managing your database server



After you're authenticated, you'll be shown the interface in Figure 1.13, which gives you many ways to interact with, and manage, your SQL Server 2005 instance.

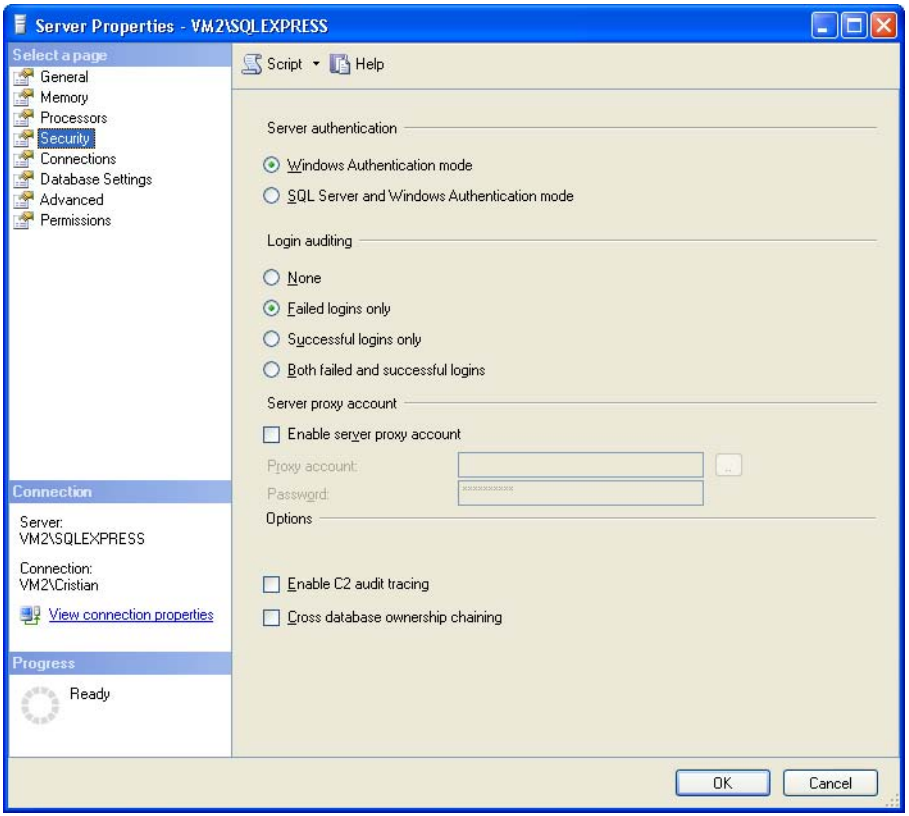
SQL Server Management Studio lets you browse through the objects inside your SQL Server, and even modify their settings. For example, you can change the security settings of your server by right-clicking the *COMPUTER*\SQLEXPRESS (where *COMPUTER* is the name of your computer), choosing Properties, and selecting Security from the panel, as shown in Figure 1.14.

SQL Server and Instances

You can run multiple SQL Servers on the one computer simultaneously—each SQL Server is called an **instance** of SQL Server. How is this useful? Imagine you have a production server that runs two applications with two separate databases on the same instance of SQL Server. If, for some reason, we need to restart SQL Server for the first application, the second application's database will become unavailable while the restart is taking place. If the second application's database was operating on a second instance of SQL Server, we wouldn't have such a problem—the second application would continue working without missing a beat.

Each instance of SQL Server requires a name. The default instance name for SQL Server 2005 Express Edition is `SQLEXPRESS`. When connecting to your

Figure 1.14. Changing server settings with SQL Server Management Studio



database server, you must specify both the name of the computer and the name of the SQL Server instance in the form *`ComputerName`*`\`*`Instance-Name`*. You can see this specification back in Figure 1.12 and Figure 1.13, where we're connecting to an instance called `SQLEXPRESS` on a computer called `VM2`.

Installing Visual Web Developer 2005

Visual Web Developer automates many of the tasks that you'd need to complete yourself in other environments, and includes many powerful features. For the first exercises in this book, we'll recommend you use a simple text editor such as

Notepad, but you'll gradually learn how to use Visual Web Developer to ease some of the tasks we'll tackle.

So let's install this tool to make sure we'll have it ready when we need it.

1. Go to http://msdn.microsoft.com/vstudio/express/vwd/ and click the **Download** link.
2. Execute the downloaded file.
3. Accept the default options. At one point, you'll be asked about installing Microsoft MSDN 2005 Express Edition, which is the product's documentation. It wouldn't hurt to install it, but you need to be patient, because it's quite big. (Note that you've already installed the .NET Framework 2.0 documentation, together with the SDK.)

Bonus!

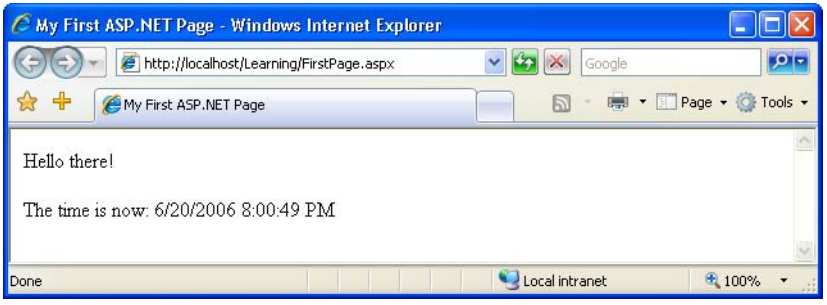
If you've already installed the .NET Framework 2.0 SDK, you've already installed Microsoft MSDN 2005 Express Edition.

In this book, we'll start using Visual Web Developer to build real web applications in Chapter 5. Until then, we'll create examples using Notepad (or another simple text editor) so you're prepared to take full advantage of the features offered by Visual Web Developer when the time comes to use it.

Writing your First ASP.NET Page

For your first ASP.NET exercise, we'll create the simple example shown in Figure 1.15.

Figure 1.15. Your first ASP.NET page



Let's get started! Open your text editor (Notepad is fine). If you have software that creates ASP.NET pages automatically, such as Visual Studio .NET or Visual Web Developer 2005 Express Edition, please don't use it yet—while these are great tools that allow you to get up and running quickly, they do assume that you already understand how ASP.NET works.

So, open your text editor, and create a new file named `FirstPage.aspx` in the `Learning` folder you created earlier. Start editing `FirstPage.aspx` by entering the HTML for our page, shown below:

File: **FirstPage.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>My First ASP.NET 2.0 Page</title>
  </head>
  <body>
    <p>Hello there!</p>
    <p>The time is now: </p>
  </body>
</html>
```

So far, so good, right? Now, let's add some ASP.NET code that will create the dynamic elements of the page, starting with the time.

File: **FirstPage.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>My First ASP.NET 2.0 Page</title>
  </head>
  <body>
    <p>Hello there!</p>
    <p>The time is now:
        <asp:Label runat="server" id="timeLabel" /></p>
  </body>
</html>
```

We've added an `<asp:Label/>` tag to the document. This is a special tag that lets us insert dynamic content into the page. The `asp:` part of the tag name identifies it as a built-in ASP.NET tag. ASP.NET comes with numerous built-in tags; `<asp:Label/>` is one of the simplest.

The `runat="server"` attribute identifies the tag as something that needs to be handled on the server. In other words, the web browser will never see the `<asp:Label/>` tag; when the page is requested by the client, ASP.NET sees it and converts it to regular HTML tags before the page is sent to the browser. It's up to us to write the code that will tell ASP.NET to replace this particular tag with the current time.

To do this, we must add some script to our page. ASP.NET gives you the choice of a number of different languages to use in your scripts. The two most common languages are VB and C#. Let's take a look at examples using both. Here's a version of the page in VB:

Visual Basic — File: **FirstPage.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>My First ASP.NET Page</title>
    <script runat="server" language="VB">
      Sub Page_Load(sender As Object, e As EventArgs)
        timeLabel.Text = DateTime.Now.ToString()
      End Sub
    </script>
  </head>
  <body>
    <p>Hello there!</p>
    <p>The time is now:
       <asp:Label runat="server" id="timeLabel" /></p>
  </body>
</html>
```

Here's the same page written in C#:

C# — File: **FirstPage.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>My First ASP.NET Page</title>
      <script runat="server" language="C#">
        protected void Page_Load(object sender, EventArgs e)
        {
          timeLabel.Text = DateTime.Now.ToString();
        }
    </script>
```

```
  </head>
  <body>
    <p>Hello there!</p>
    <p>The time is now:
       <asp:Label runat="server" id="timeLabel" /></p>
  </body>
</html>
```

Clues for Case Sensitivity

Note that C#, unlike VB, is case-sensitive. If you type the case of a letter incorrectly, the page won't load. If these languages look complicated, don't worry: you'll learn more about them in Chapter 3.

Both versions of the page achieve exactly the same thing. You can even save them both, giving each a different filename, and test them separately. If you've never done any server-side programming before, the code may look a little scary. But before we analyze it in detail, let's load the page and test that it works. Using your web browser, load `http://localhost/Learning/FirstPage.aspx`. Whether you load the C# version or the VB version, the output should look like Figure 1.15.

No Time?

If the time isn't displayed in the page, chances are that you opened the file directly in your browser instead of loading it through your web server. Because ASP.NET is a server-side language, your web server needs to process the file before it's sent to your browser for display. If it doesn't get access to the file, the ASP.NET code is never converted into HTML that your browser can understand, so make sure you load the page by entering an actual HTTP URL (such as `http://localhost/Learning/FirstPage.aspx`), not a local path and filename (such as `C:\WebDocs\Learning\FirstPage.aspx`).

What happens there? Let's break down the new elements of this page.

File: **FirstPage.aspx (excerpt)**

```
<script runat="server">
```

This tag marks the start of server-side code, or the **code declaration block**. Like the `<asp:Label/>` tag, this `<script>` tag uses the `runat="server"` attribute to let ASP.NET know that the tag should be processed before sending the page to the browser.

Visual Basic File: **FirstPage.aspx (excerpt)**

```
Sub Page_Load(sender As Object, e As EventArgs)
```

C# File: **FirstPage.aspx (excerpt)**

```
protected void Page_Load(object sender, EventArgs e)
{
```

I won't go into too much detail here. For now, all you need to know is that you can write script fragments that are run in response to different events, such as a button being clicked or an item being selected from a drop-down list. What the first line of code basically says is, "execute the following script whenever the page is loaded." Note that C# groups code into blocks with curly braces (`{` and `}`), while Visual Basic uses statements such as `End Sub` to mark the end of a particular sequence. So, the curly brace in the C# code above (`{`) marks the start of the script that will be executed when the page loads for the first time.

Finally, here's the line that actually displays the time on the page:

Visual Basic File: **FirstPage.aspx (excerpt)**

```
  timeLabel.Text = DateTime.Now.ToString()
```

C# File: **FirstPage.aspx (excerpt)**

```
  timeLabel.Text = DateTime.Now.ToString();
```

As you can see, these .NET languages have much in common, because they're both built on the .NET Framework. In fact, the only difference between the ways the two languages handle the above line is that C# ends lines of code with a semicolon (`;`). In plain English, here's what this line says:

> Set the `Text` of `timeLabel` to the current date and time, expressed as text.

Note that `timeLabel` is the value we gave for the `id` attribute of the `<asp:Label/>` tag where we want to show the time. So, `timeLabel.Text`, or the `Text` property of `timeLabel`, refers to the text that will be displayed by the tag. `DateTime` is a **class** that's built into the .NET Framework; it lets you perform all sorts of useful functions with dates and times. The .NET Framework has thousands of these classes, which do countless handy things. The classes are collectively known as the **.NET Framework Class Library**.

The `DateTime` class has a **property** called `Now`, which returns the current date and time. This `Now` property has a **method** called `ToString`, which expresses that date and time as text (a segment of text is called a **string** in programming circles).

Classes, properties, and methods: these are all important words in the vocabulary of any programmer, and we'll discuss them in more detail a little later in the book. For now, all you need to take away from this discussion is that `DateTime.Now.ToString()` will give you the current date and time as a text string, which you can then tell your `<asp:Label/>` tag to display. The rest of the script block simply ties up loose ends:

Visual Basic — File: **FirstPage.aspx (excerpt)**

```
End Sub
</script>
```

C# — File: **FirstPage.aspx (excerpt)**

```
}
</script>
```

The `End Sub` in the VB code, and the `}` in the C# code, mark the end of the script that's to be run when the page is loaded, and the `</script>` tag marks the end of the script block.

One final thing that's worth investigating is the code that ASP.NET generated for you. It's clear by now that your web browser receives only HTML (no server-side code!), so what kind of HTML was generated for that label? The answer is easy to find! With the page displayed in your browser, you can use the browser's View Source feature to view the page's HTML code. Here's what you'll see:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>My First ASP.NET Page</title>
  </head>
  <body>
    <p>Hello there!</p>
    <p>The time is now:
       <span id="timeLabel">6/20/2006 8:00:49 PM</span></p>
  </body>
</html>
```

Notice that all the ASP.NET code has gone! Even the script block has been completely removed, and the `<asp:Label/>` tag has been replaced by a `<span>` tag (which has the same `id` attribute as the `<asp:Label/>` tag we used) that contains the date and time.

That's how ASP.NET works. From the web browser's point of view, there's nothing special about an ASP.NET page; it's just plain HTML like any other. All the ASP.NET code is run by your web server and converted to plain HTML that's sent to the browser. So far, so good: the example above was fairly simple. The next chapter will get a bit more challenging as we investigate some valuable programming concepts.

Getting Help

As you develop ASP.NET web applications, you will undoubtedly have questions that need answers, and problems that need to be solved. The ASP.NET support web site[7] was developed by Microsoft as a portal for the ASP.NET community to answer the questions and solve the problems that developers encounter while using ASP.NET. The support web site provides useful information, such as news, downloads, articles, and discussion forums. You can also ask questions of the experienced community members in the SitePoint Forums.[8]

Summary

In this chapter, you learned about .NET. You also learned of the benefits of ASP.NET and that it's a part of the .NET Framework.

First, you learned about the components of ASP.NET and how to locate and install the .NET Framework. Then, we explored the software that's required not only to use this book, but also in order for you or your company to progress with ASP.NET development.

You've gained a solid foundation in the world of ASP.NET! The next chapter will build on this knowledge as we begin to introduce you to ASP.NET in more detail, covering page structure, the languages that you can use, various programming concepts, and the finer points of form processing.

[7] http://www.asp.net/
[8] http://www.sitepoint.com/forums/

2 ASP.NET Basics

So far, you've learned what ASP.NET is, and what it can do. You've installed the software you need to get going, and, having been introduced to some very simple form processing techniques, you even know how to create a simple ASP.NET page. Don't worry if it all seems a little bewildering right now, because, as this book progresses, you'll learn how to use ASP.NET at more advanced levels.

As the next few chapters unfold, we'll explore some more advanced topics, including the use of controls, and various programming techniques. But before you can begin to develop applications with ASP.NET, you'll need to understand the inner workings of a typical ASP.NET page—with this knowledge, you'll be able to identify the parts of the ASP.NET page referenced in the examples we'll discuss throughout this book. So, in this chapter, we'll talk about some key mechanisms of an ASP.NET page, specifically:

- page structure
- view state
- namespaces
- directives

We'll also cover two of the "built-in" languages supported by the .NET Framework: VB and C#. As this section progresses, we'll explore the differences and similarities between these two languages, and get a clear idea of the power that they provide to those creating ASP.NET applications.

So, what exactly makes up an ASP.NET page? The next few sections will give you an in-depth understanding of the constructs of a typical ASP.NET page.

ASP.NET Page Structure

ASP.NET pages are simply text files that have the `.aspx` file name extension, and can be placed on any web server equipped with ASP.NET.

When a client requests an ASP.NET page, the web server passes the page to the **ASP.NET runtime**, a program that runs on the web server that's responsible for reading the page and compiling it into a .NET class. This class is then used to produce the HTML that's sent back to the user. Each subsequent request for this page avoids the compilation process: the .NET class can respond directly to the request, producing the page's HTML and sending it to the client, until such time as the `.aspx` file changes. This process is illustrated in Figure 2.1.

An ASP.NET page consists of the following elements:

- directives
- code declaration blocks
- code render blocks
- ASP.NET server controls
- server-side comments
- literal text and HTML tags

Figure 2.2 illustrates the various parts of a simple ASP.NET page.

Figure 2.1. The life cycle of the ASP.NET page

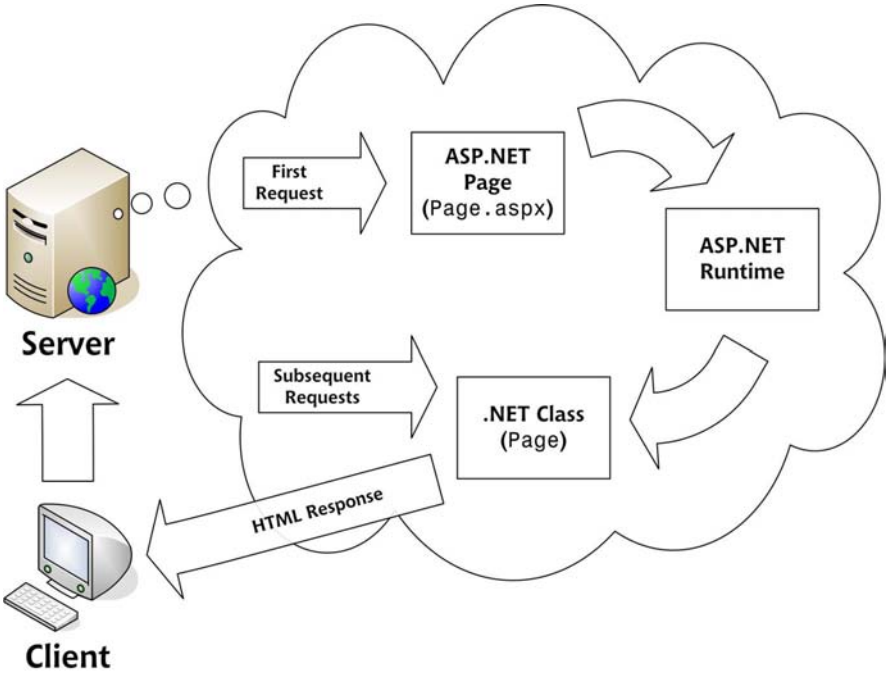
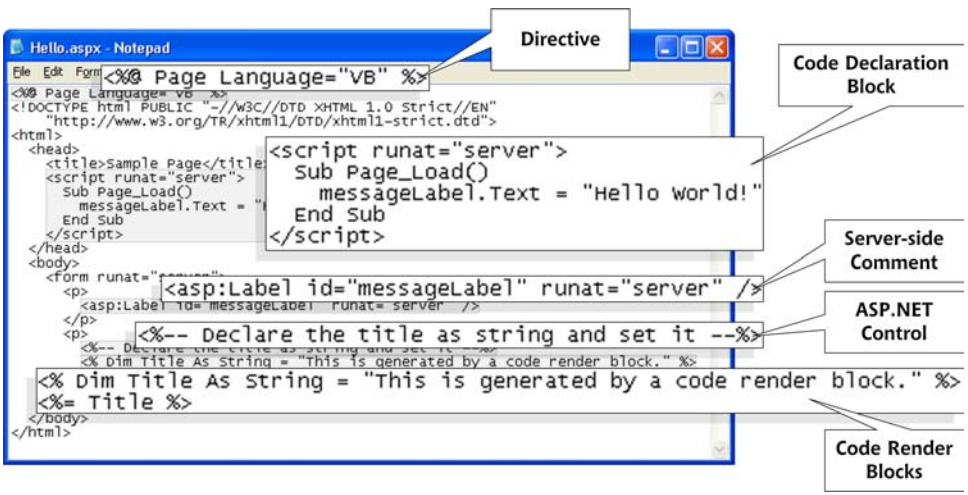


Figure 2.2. The parts of an ASP.NET page



To make sure we're on the same page and that the code works, save this piece of code in a file named `Hello.aspx` within the `Learning` virtual directory you created in Chapter 1. Loading the file through `http://localhost/Learning/Hello.aspx` should render the result shown in Figure 2.3.

Figure 2.3. Sample Page in action

As you can see, this ASP.NET page contains examples of all the above components (except server-side includes) that make up an ASP.NET page. You won't often use every single element in a given page, but it's important that you are familiar with these elements, their purposes, and how and when it's appropriate to use them.

Directives

The directives section is one of the most important parts of an ASP.NET page. Directives control how a page is compiled, specify how a page is cached by web browsers, aid debugging (error-fixing), and allow you to import classes to use within your page's code. Each directive starts with `<%@`. This is followed by the directive name, plus any attributes and their corresponding values. The directive then ends with `%>`.

There are many directives that you can use within your pages, and we'll discuss them in greater detail later, but, for now, know that the `Import` and `Page` directives are the most useful for ASP.NET development. Looking at the sample ASP.NET page in Figure 2.2, we can see that a `Page` directive was used at the top of the page like so:

Visual Basic — File: **`Hello.aspx` (excerpt)**

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

C# File: **Hello.aspx (excerpt)**

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

In this case, the `Page` directive specifies the language that's to be used for the application logic by setting the `Language` attribute. The value provided for this attribute, which appears in quotes, specifies that we're using either VB or C#. A whole range of different directives is available; we'll see a few more later in this chapter.

Unlike ASP, ASP.NET directives can appear anywhere on a page, but they're commonly included at its very beginning.

Code Declaration Blocks

In Chapter 3, we'll talk about code-behind pages and how they let us separate our application logic from an ASP.NET page's HTML. However, if you're not working with code-behind pages, you must use code declaration blocks to contain all the application logic of your ASP.NET page. This application logic defines variables, subroutines, functions, and more. In our page, we've placed the code inside `<script>` tags, like so:

Visual Basic

```
<script runat="server">
  Sub mySub()
    ' Code here
  End Sub
</script>
```

Here, the tags enclose VB code, but it could just as easily be C#:

C#

```
<script runat="server">
  void mySub()
  {
    // Code here
  }
</script>
```

Comments in VB and C# Code

Both of these code snippets contain comments—explanatory text that will be ignored by ASP.NET, but which serves to describe to us how the code works.

In VB code, a single quote or apostrophe (`'`) indicates that the remainder of the line is to be ignored as a comment.

In C# code, two slashes (`//`) achieve the same end. C# code also lets us span a comment over multiple lines if we begin it with `/*` and end it with `*/`, as in this example:

C#

```
<script runat="server">
  void mySub()
  {
    /* Multi-line
       comment    */
  }
</script>
```

Before .NET emerged, ASP also supported such script tags using a `runat="server"` attribute. However, they could only ever contain VBScript and, for a variety of reasons, they failed to find favor among developers.

Code declaration blocks are generally placed inside the `head` of your ASP.NET page. The sample ASP.NET page shown in Figure 2.2, for instance, contains the following code declaration block:

Visual Basic File: **Hello.aspx (excerpt)**

```
<script runat="server">
  Sub Page_Load()
    messageLabel.Text = "Hello World"
  End Sub
</script>
```

Perhaps you can work out what the equivalent C# code would be:

C# File: **Hello.aspx (excerpt)**

```
<script runat="server">
  void Page_Load()
  {
    messageLabel.Text = "Hello World";
  }
</script>
```

The `<script runat="server">` tag also accepts two other attributes. We can set the language that's used in this code declaration block via the `language` attribute:

Visual Basic
```
<script runat="server" language="VB">
```

C#
```
<script runat="server" language="C#">
```

If you don't specify a language within the code declaration block, the ASP.NET page will use the language provided by the `language` attribute of the `Page` directive. Each page's code must be written in a single language; for instance, it's not possible to mix VB and C# in the same page.

The second attribute that's available to us is `src`; this lets us specify an external code file for use within the ASP.NET page:

Visual Basic
```
<script runat="server" language="VB" src="mycodefile.vb">
```

C#
```
<script runat="server" language="C#" src="mycodefile.cs">
```

Code Render Blocks

If you've had experience with traditional ASP, you might recognize these blocks. You can use code render blocks to define inline code or expressions that will execute when a page is rendered. Code within a code render block is executed immediately when it is encountered—usually when the page is loaded or rendered. On the other hand, code within a code *declaration* block (within `<script>` tags) is executed only when it is called or triggered by user or page interactions. There are two types of code render blocks—inline code, and inline expressions—both of which are typically written within the body of the ASP.NET page.

Inline code render blocks execute one or more statements, and are placed directly inside a page's HTML between `<%` and `%>` delimiters. In our example, the following is a code render block:

Visual Basic — File: **`Hello.aspx` (excerpt)**
```
<% Dim Title As String = "This is generated by a code render " & _
    "block." %>
```

C# — File: **`Hello.aspx` (excerpt)**
```
<% string Title = "This is generated by a code render block."; %>
```

These code blocks simply declare a String variable called `Title`, and assign it the value `This is generated by a code render block`.

Inline expression render blocks can be compared to `Response.Write` in classic ASP. They start with `<%=` and end with `%>`, and are used to display the values of variables and methods on a page. In our example, an inline expression appears immediately after our inline code block:

File: **Hello.aspx (excerpt)**

```
<%= Title %>
```

If you're familiar with classic ASP, you'll know what this code does: it simply outputs the value of the variable `Title` that we declared in the previous inline code block.

ASP.NET Server Controls

At the heart of any ASP.NET page lie server controls, which represent dynamic elements with which your users can interact. There are three basic types of server control: ASP.NET controls, HTML controls, and web user controls.

Usually, an ASP.NET control must reside within a `<form runat="server">` tag in order to function correctly. Controls offer the following advantages to ASP.NET developers:

- They give us the ability to access HTML elements easily from within our code: we can change these elements' characteristics, check their values, or even update them dynamically from our server-side programming language of choice.
- ASP.NET controls retain their properties thanks to a mechanism called **view state**. We'll be covering view state later in this chapter. For now, you need to know that view state prevents users from losing the data they've entered into a form once that form has been sent to the server for processing. When the response comes back to the client, text box entries, drop-down list selections, and so on, are all retained through view state.
- With ASP.NET controls, developers are able to separate a page's presentational elements (everything the user sees) from its application logic (the dynamic portions of the ASP.NET page), so that each can be considered separately.
- Many ASP.NET controls can be "bound" to the data sources from which they will extract data for display with minimal (if any) coding effort.

ASP.NET is all about controls, so we'll be discussing them in greater detail as we move through this book. In particular, Chapter 4 explains many of the controls that ship with ASP.NET. For now, though, let's continue with our dissection of an ASP.NET page.

Server-side Comments

Server-side comments allow you to include within the page comments or notes that will not be processed by ASP.NET. Traditional HTML uses the `<!--` and `-->` character sequences to delimit comments; any information included between these tags will not be displayed to the user. ASP.NET comments look very similar, but use the sequences `<%--` and `--%>`.

Our ASP.NET example contains the following server-side comment block:

File: **Hello.aspx (excerpt)**

```
<%-- Declare the title as string and set it --%>
```

The difference between ASP.NET comments and HTML comments is that ASP.NET comments are not sent to the client at all; HTML comments are, so they're not suited to commenting out ASP.NET code. Consider the following example:

C#

```
<!--
<% string Title = "This is generated by a code render block."; %>
<%= Title %>
-->
```

Here, it looks as if a developer has attempted to use an HTML comment to stop a code render block from being executed. Unfortunately, HTML comments will only hide information from the browser, not the ASP.NET runtime. So, in this case, while we won't see anything in the browser that represents these two lines, they will be processed by ASP.NET, and the value of the variable `Title` will be sent to the browser inside an HTML comment, as shown here:

```
<!—-

This code generated by a code render block.
-->
```

The code could be modified to use server-side comments very simply:

C#

```
<%--
<% string Title = "This is generated by a code render block."; %>
<%= Title %>
--%>
```

The ASP.NET runtime will ignore the contents of this comment, and the value of the `Title` variable will not be output.

Literal Text and HTML Tags

The final elements of an ASP.NET page are plain old text and HTML. Generally, you can't do without these elements—after all, HTML allows the display of the information in your ASP.NET controls and code in a way that's suitable for users. Let's take a look at the literal text and HTML tags that were used to produce the display in Figure 2.2:

Visual Basic File: **`Hello.aspx` (excerpt)**

```
<%@ Page Language="VB" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page</title>
    <script runat="server">
      Sub Page_Load()
        messageLabel.Text = "Hello World!"
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <p>
        <asp:Label id="messageLabel" runat="server" />
      </p>
      <p>
        <%-- Declare the title as string and set it --%>
        <% Dim Title As String = "This is generated by a " & _
            "code render block." %>
        <%= Title %>
      </p>
    </form>
  </body>
</html>
```

C# File: **Hello.aspx (excerpt)**

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page</title>
    <script runat="server">
      void Page_Load()
      {
        messageLabel.Text = "Hello World";
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <p>
        <asp:Label id="messageLabel" runat="server" />
      </p>
      <p>
        <%-- Declare the title as string and set it --%>
        <% string Title = "This is generated by a code render " +
            "block."; %>
        <%= Title %>
      </p>
    </form>
  </body>
</html>
```

The bold code above highlights the fact that literal text and HTML tags provide the structure for presenting our dynamic data. Without these elements, this page would have no format, and the browser would be unable to understand it.

You now have a clearer understanding of the structure of an ASP.NET page. As you work through the examples in this book, you'll begin to realize that, in many cases, you won't need to use all of these elements. For the most part, your development will be modularized within code declaration blocks, and all of the dynamic portions of your pages will be contained within code render blocks or controls located inside a `<form runat="server">>` tag.

In the following sections, we'll explore view state, discuss working with directives, and shine a little light on the languages that can be used within ASP.NET.

View State

As we saw briefly in the previous section, ASP.NET controls automatically retain their data when a page is sent to the server in response to an event (such as a user clicking a button). Microsoft calls this persistence of data "view state." In the past, developers would have had to use hacks to remember the item a user had selected in a drop-down menu, or store the content entered into a text box; typically, these hacks would have relied on hidden form fields.

This is no longer the case: once they're submitted to the server for processing, ASP.NET pages automatically retain all the information contained in text boxes and drop-down lists, as well as radio button and checkbox selections. They even keep track of dynamically generated tags, controls, and text. Consider the following ASP (not ASP.NET!) page, called `Sample.asp`:

File: **Sample.asp**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page using VBScript</title>
  </head>
  <body>
    <form method="post" action="sample.asp">
      <input type="text" name="nameTextBox"/>
      <input type="submit" name="submitButton"
          value="Click Me" />
      <%
      If Request.Form("nameTextBox") <> "" Then
        Response.Write(Request.Form("nameTextBox"))
      End If
      %>
    </form>
  </body>
</html>
```

Cassini and ASP

Cassini is an ASP.NET-only web server and will not execute pages written in ASP, such as `Sample.asp` above. Fortunately, this won't be a problem as you work your way through this book, as the above `Sample.asp` file is the only ASP code in this book.

If you save this as `Sample.asp` in the `Learning` virtual directory you created in Chapter 1, and open it in your browser by entering **`http://localhost/Learning/Sample.asp`**, you'll see that view state is not automatically preserved. When the user submits the form, the information that was typed into the text box is cleared, although it's still available in `Request.Form("nameTextBox")`. The equivalent page in ASP.NET, `ViewState.aspx`, demonstrates this data persistence using view state:

Visual Basic — File: **`ViewState.aspx`**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>View State Example</title>
    <script runat="server" language="VB">
      Sub Click(s As Object, e As EventArgs)
        messageLabel.Text = nameTextBox.Text
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:TextBox id="nameTextBox" runat="server" />
      <asp:Button id="submitButton" runat="server"
          Text="Click Me" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

C# — File: **`ViewState.aspx`**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>View State Example</title>
    <script runat="server" language="C#">
      void Click(Object s, EventArgs e)
      {
        messageLabel.Text = nameTextBox.Text;
      }
    </script>
  </head>
  <body>
    <form runat="server">
```

```
      <asp:TextBox id="nameTextBox" runat="server" />
      <asp:Button id="submitButton" runat="server"
          Text="Click Me" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

In this case, the code uses ASP.NET controls with the `runat="server"` attribute. As you can see in Figure 2.4, the text from the box appears on the page when the button is clicked, but also notice that the data remains in the text box! The data in this example is preserved by view state.

Figure 2.4. ASP.NET maintaining the state of the controls

You can see the benefits of view state already. But where's all that information stored? ASP.NET pages maintain view state by encrypting the data within a hidden form field. View the source of the page after you've submitted the form, and look for the following code:

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
    value="/wEPDwUKLTEwNDY1NzgOMQ9…OfMCR+FN5P6v5pkTQwNEl5xhBk" />
```

This is a standard HTML hidden form field. All information that's relevant to the view state of the page is stored within this hidden form field as an encrypted string.

View state is enabled for every page by default. If you do not intend to use view state, you can turn it off, which will result in a slight performance gain in your pages. To do this, set the `EnableViewState` property of the `Page` directive to `false`:

```
<%@ Page EnableViewState="False" %>
```

Disabling View State, Control by Control

View state can also be disabled for particular controls in a page: simply set their `EnableViewState` property to `false`. We'll see working examples of this in the following chapters.

Speaking of directives, it's time to take a closer look at these curious beasts!

Working with Directives

For the most part, ASP.NET pages resemble traditional HTML pages with a few additions. In essence, just using the `.aspx` extension for an HTML file will ensure that IIS passes the page to the .NET Framework for processing. However, before you can work with certain, more advanced features, you'll need to know how to use directives.

We talked a little about directives and what they can do earlier in this chapter. You learned that directives control how a page is created, how a page is cached, help with bug-fixing, and allow us to import advanced functionality for use within our code. Three of the most commonly used directives are:

`Page`
: This directive defines page-specific attributes for the ASP.NET page, such as the language used for server-side code. We've already seen this directive in use.

`Import`
: The `Import` directive makes functionality that's been defined elsewhere available in a given page. The following example, for instance, imports functionality from the `System.Web.Mail` namespace, which you could use to send email from a page. Namespaces are simply .NET's way of keeping all its functionality neatly organized—we'll see how they work in Chapter 3.

```
<%@ Import Namespace="System.Web.Mail" %>
```

You'll become very familiar with this directive as you work through this book.

`Register`
: This directive allows you to register a user control for use on your page. We'll cover `Register` in detail in Chapter 4, but the directive looks something like this:

```
<%@ Register TagPrefix="uc" TagName="footer"
    Src="footer.ascx" %>
```

ASP.NET Languages

As we saw in the previous chapter, .NET supports many different languages; in fact, there's no limit to the number of languages that could be supported. If you're used to writing ASP, you may think the choice of VBScript or JScript would be an obvious one. But, with ASP.NET, Microsoft did away with VBScript, merging it with Visual Basic. ASP.NET's support for C# is likely to find favor with developers from other backgrounds. This section will introduce you to both these new languages, which will be covered in more depth in the next chapter. By the end of this section, you will, I hope, agree that the similarities between the two are astonishing—any differences are minor and, in most cases, easy to figure out.

Traditional server technologies are much more constrained in terms of the development languages they offer. For instance, old-style CGI scripts were typically written with Perl or C/C++, JSP uses Java, Coldfusion uses CFML, and PHP is a technology and a language rolled into one. .NET's support for many different languages lets developers choose the ones they prefer. To keep things simple, this book will consider the two most popular: VB and C#. You can choose the language that feels more comfortable to you, or stick with your current favorite if you have one.

Visual Basic

The latest version of Visual Basic is the result of a dramatic overhaul of Microsoft's hugely popular Visual Basic language. With the inception of Rapid Application Development (RAD) in the 1990s, Visual Basic became extremely popular, allowing in-house teams and software development shops to bang out applications two-to-the-dozen. The latest version of VB has many advantages over older versions, most notably the fact that it has now became a fully object oriented language. At last, it can call itself a true programming language that's on a par with the likes of Java and C++. Despite the changes, VB generally stays close to the structured, legible syntax that has always made it so easy to read, use, and maintain.

C#

The official line is that Microsoft created C# in an attempt to produce a programming language that coupled the simplicity of Visual Basic with the power and flexibility of C++. However, there's little doubt that its development was at least hurried along by Microsoft's legal disputes with Sun. After Microsoft's treatment (some would say abuse) of Sun's Java programming language, Microsoft was forced to stop developing its own version of Java, and instead developed C# and another language, which it calls J#. We're not going to worry about J# here, as C# is preferable. It's easy to read, use, and maintain, because it does away with much of the confusing syntax for which C++ became infamous.

Summary

In this chapter, we started out by introducing key aspects of an ASP.NET page including directives, code declaration blocks, code render blocks, includes, comments, and controls. As the chapter progressed, we took a closer look at the two most popular languages that ASP.NET supports, and which we'll use throughout this book.

In the next chapter, we'll create a few more ASP.NET pages to demonstrate form processing techniques and programming basics, before we turn our attention to the topic of object oriented programming for the Web.

3

VB and C# Programming Basics

As you learned at the end of the last chapter, one of the great things about using ASP.NET is that we can pick and choose which of the various .NET languages we like. In this chapter, we'll look at the key programming principles that will underpin our use of Visual Basic and C#. We'll start by discussing some basic concepts of programming ASP.NET web applications using these two languages. We'll explore programming fundamentals such as variables, arrays, functions, operators, conditionals, loops, and events, and work through a quick introduction to object oriented programming (OOP). Next, we'll dive into namespaces and address the topic of classes—seeing how they're exposed through namespaces, and which ones you'll use most often.

The final sections of the chapter cover some of the ideas underlying modern, effective ASP.NET design, including code-behind and the value it provides by helping us separate code from presentation. We finish with an examination of how object oriented programming techniques impact the ASP.NET developer.

Programming Basics

One of the building blocks of an ASP.NET page is the application logic: the actual programming code that allows the page to function. To get anywhere with ASP.NET, you need to grasp the concept of **events**. All ASP.NET pages will contain controls such as text boxes, checkboxes, and lists. Each of these controls

allows the user to interact with the application in some way: checking checkboxes, scrolling through lists, selecting list items, and so on. Whenever one of these actions is performed, the control will raise an event. It is by handling these events within our code that we get ASP.NET pages to do what we want.

For instance, imagine that a user clicks a button on an ASP.NET page. That button (or, more specifically, the ASP.NET `Button` control) raises an event (in this case, it will be the `Click` event). A method called an **event handler** executes automatically when an event is raised—in this case, the event handler code performs a specific action for that button. For instance, the `Click` event handler could save form data to a file, or retrieve requested information from a database. Events really are the key to ASP.NET programming, which is why we'll start this chapter by taking a closer look at them.

It wouldn't be practical, or even necessary, to cover *all* aspects of VB and C# in this book, so we're going to discuss enough to get you started, and complete this chapter's projects and samples using both languages. Moreover, we'd say that the programming concepts you'll learn here will be more than adequate to complete the great majority of day-to-day web development tasks using ASP.NET.

Control Events and Subroutines

As I just mentioned, an event (sometimes more than one) is raised, and handler code is called, in response to a specific action on a particular control. For instance, the code below creates a server-side button and label. Note the use of the `OnClick` attribute on the `Button` control. If you want to test the code, save the file in the `Learning` virtual directory you've been using for the other examples.

File: **ClickEvent.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Click the Button</title>
  </head>
  <body>
    <form runat="server">
      <asp:Button id="button" runat="server"
          OnClick="button_Click" Text="Click Me" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

When the button's clicked, it raises the `Click` event, and ASP.NET checks the button's `OnClick` attribute to find the name of the handler subroutine for that event. In the code above, we told ASP.NET to call the `button_Click` routine, so perhaps we'd better write this subroutine! We'd normally place it within a code declaration block inside the `<head>` tag, like this:

Visual Basic File: **ClickEvent.aspx (excerpt)**

```
<head>
  <title>Click the Button</title>
  <script runat="server" language="VB">
    Public Sub button_Click(s As Object, e As EventArgs)
      messageLabel.Text = "Hello World"
    End Sub
  </script>
</head>
```

C# File: **ClickEvent.aspx (excerpt)**

```
<head>
  <title>Click the Button</title>
  <script runat="server" language="C#">
    public void button_Click(Object s, EventArgs e)
    {
      messageLabel.Text = "Hello World";
    }
  </script>
</head>
```

This code simply sets a message to display on the label that we declared with the button. So, when this page is run, and users click the button, they'll see the message "Hello World" appear next to it.

Figure 3.1. Handling the Click event

Hopefully, you're starting to come to grips with the idea of control events, and the ways in which they're used to call particular subroutines. In fact, there are many events that your controls can use, though some of them are found only on

certain controls. Here's the complete set of attributes that the `Button` control supports for handling events:

`OnClick`
: As we've seen, the subroutine indicated by this attribute is called for the `Click` event, which occurs when the user clicks the button.

`OnCommand`
: As with `OnClick`, the subroutine indicated by this attribute is called when the button is clicked.

`OnLoad`
: The subroutine indicated by this attribute is called when the button is loaded for the first time—usually when the page first loads.

`OnInit`
: When the button is initialized, any subroutine given in this attribute will be called.

`OnPreRender`
: We can use this attribute to run code just before the button is rendered.

`OnUnload`
: This subroutine will run when the control is unloaded from memory—basically, when the user goes to a different page or closes the browser entirely.

`OnDisposed`
: The subroutine specified by this attribute is executed when the button is released from memory.

`OnDataBinding`
: This attribute fires when the button is bound to a data source.

Don't worry too much about the intricacies of all these events and when they occur; I just want you to understand that a single control can produce a number of different events. In the case of the `Button` control, you'll almost always be interested in the `Click` event; the others are only useful in rather obscure circumstances.

When a control raises an event, the specified subroutine (if one is specified) is executed. Let's take a look at the structure of a typical subroutine that interacts with a web control:

Visual Basic

```
Public Sub mySubName(s As Object, e As EventArgs)
  ' Write your code here
End Sub
```

C#

```
public void mySubName(Object s, EventArgs e)
{
  // Write your code here
}
```

Let's take a moment to break down all the components that make up a typical subroutine.

`Public` (Visual Basic)
`public` (C#)

This command defines the scope of the subroutine. There are a few different options to choose from, the most frequently used being `Public` (for a global subroutine that can be used anywhere within the entire page) and `Private` (for subroutines that are available for the specific class only).[1] We'll analyze these options in more detail a bit later in the chapter.

`Sub` (Visual Basic)
`void` (C#)

This command defines the chunk of code as a subroutine. A subroutine is a named block of code that doesn't return a result; thus, in C#, we use the `void` keyword, which means exactly what the name says. We don't need this in VB, though, because the `Sub` keyword implies that no value is returned.

`mySubName(…)`

This part gives the name we've chosen for the subroutine. The parameters and their data types are mentioned in the parentheses.

`s As Object` (Visual Basic)
`Object s` (C#)

When we write a subroutine that will function as an event handler, it must accept two parameters. The first is a reference to the control that fired the event. Each control has a particular type, such as `Label` or `TextBox`, but `Object` is a generic type that can be used to reference any kind of object in .NET—even basic types, such as numbers or strings. Here, we're putting that `Object` in a variable named `s` (again, we'll talk more about variables later in

[1] The C# equivalents of `Public` and `Private` are, perhaps predictably, `public` and `private`.

this chapter). We can then use that variable to access features and settings of the specific control from our subroutine.

`e As EventArgs` (Visual Basic)
`EventArgs e` (C#)
This, the second parameter, contains certain information that's specific to the event that was raised. Note that, in many cases, you won't need to use either of these two parameters, so you don't need to worry about them too much at this stage.

As this chapter progresses, you'll see how subroutines that are associated with particular events by the appropriate attributes on controls can revolutionize the way your user interacts with your application.

Page Events

Until now, we've considered only events that are raised by controls. However, there is another type of event: the page event. Technically, a page is simply another type of control, so page events are a particular kind of control event.

The idea is the same as for control events, except that here, it is the page as a whole that generates the events.[2] You've already used one of these events: the `Page_Load` event, which is fired when the page loads for the first time. Note that we don't need to associate handlers for page events as we did for control events; instead, we just place our handler code inside a subroutine with a preset name.

The following list outlines the most frequently used page event subroutines:

`Page_Init`
called when the page is about to be initialized with its basic settings

`Page_Load`
called once the browser request has been processed, and all of the controls in the page have their updated values

`Page_PreRender`
called once all objects have reacted to the browser request and any resulting events, but before any response has been sent to the browser

[2] Strictly speaking, a page is simply another type of control, so page events *are* actually control events. But when you're first learning ASP.NET, it can be helpful to think of page events as being different, especially since you don't usually use `On`*`EventName`* attributes to assign subroutines to handle them.

`Page_UnLoad`
: called when the page is no longer needed by the server, and is ready to be discarded

The order in which the events are listed above is also the order in which they're executed. In other words, the `Page_Init` event is the first event raised by the page, followed by `Page_Load`, `Page_PreRender`, and finally `Page_UnLoad`.

The best way to illustrate how these events work is through an example. Create the following `PageEvents.aspx` file in the `Learning` virtual directory:

Visual Basic — File: **`PageEvents.aspx` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Page Events</title>
    <script runat="server" language="VB">
      Sub Page_Init(s As Object, e As EventArgs)
        messageLabel.Text = "1. Page_Init <br/>"
      End Sub
      Sub Page_Load(s As Object, e As EventArgs)
        messageLabel.Text += "2. Page_Load <br/>"
      End Sub
      Sub Page_PreRender(s As Object, e As EventArgs)
        messageLabel.Text += "3. Page_PreRender <br/>"
      End Sub
      Sub Page_UnLoad(s As Object, e As EventArgs)
        messageLabel.Text += "4. Page_UnLoad <br/>"
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

C# — File: **`PageEvents.aspx` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Page Events</title>
```

```
    <script runat="server" language="C#">
      void Page_Init(Object s, EventArgs e)
      {
        messageLabel.Text = "1. Page_Init <br/>";
      }
      void Page_Load(Object s, EventArgs e)
      {
        messageLabel.Text += "2. Page_Load <br/>";
      }
      void Page_PreRender(Object s, EventArgs e)
      {
        messageLabel.Text += "3. Page_PreRender <br/>";
      }
      void Page_UnLoad(Object s, EventArgs e)
      {
        messageLabel.Text += "4. Page_UnLoad <br/>";
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

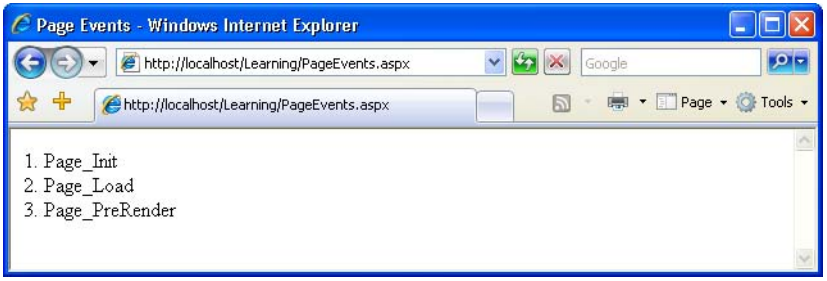
You can see that the event handlers (the functions that are executed to handle the events) aren't specifically defined anywhere. There's no need to define them, because these events are generated by default by the ASP.NET page, and their handlers have the default names that we've used in the code (`Page_Init`, `Page_Load`, etc). As the page loads, it will generate a number of events. Within each event's event handler, we've added a message to the `Label` control; this will give us visual proof that the events actually fire in order. No matter which version of the code you execute (C# or VB), the output should look like Figure 3.2.

As you can see, `Page_UnLoad` doesn't generate any output. This is because, at that point, the HTML output has already been generated and sent to the browser.

Popular Page_Load

The event you'll make the most use of in your code is `Page_Load`. However, in certain situations the other events will be helpful as well. It's also worth noting that ASP.NET supports other events, which we haven't covered here. You'll only need those in certain, complex applications that aren't in the scope of this book.

Figure 3.2. Handling ASP.NET events



Variables and Variable Declaration

Variables are fundamental to programming, and you're almost certain to have come across the term before. Basically, variables let you give a name, or identifier, to a specific piece of data; we can then use that identifier to store, modify, and retrieve the data in question.

VB and C# have access to the same basic data types, which are defined as foundation classes of the .NET Framework. However, they can be named differently, as each language defines its own aliases. There are many different kinds of data types, including strings, integers (whole numbers), and floating point numbers (fractions or decimals). Before you can use a variable in VB or C#, you must specify the types of data it can contain, using keywords such as `String`, `Integer`, and `Decimal`, like this:

Visual Basic

```
Dim name As String
Dim age As Integer
```

C#

```
string name;
int age;
```

These lines declare the type of data we want our variables to store, and are therefore known as "variable declarations." In VB, we use the keyword `Dim`, which stands for "dimension," while in C#, we simply precede the variable name with the appropriate data type.

Sometimes, we want to set an initial value for variables that we declare; we can do this using a process known as initialization:

Visual Basic
```
Dim carType As String = "BMW"
```

C#
```
string carType = "BMW";
```

We can also declare and/or initialize a group of variables of the same type simultaneously. This practice isn't recommended, though, as it makes the code more difficult to read.

Visual Basic
```
Dim carType As String, carColor As String = "blue"
```

C#
```
string carType, carColor = "blue";
```

Table 3.1 lists the most useful data types available in VB and C#.

Table 3.1. A list of commonly used data types

VB	C#	Description
`Integer`	`int`	whole numbers in the range -2,147,483,648 to 2,147,483,647
`Decimal`	`decimal`	numbers up to 28 decimal places; this command is used most often when dealing with costs of items
`String`	`string`	any text value
`Char`	`char`	a single character (letter, number, or symbol)
`Boolean`	`bool`	true or false
`Object`	object	a generic type that can be used to refer to objects of any type

You'll encounter many other data types as you progress, but this list provides an overview of the ones you'll use most often.

Many Aliases are Available

These data types are the VB- and C#-specific aliases for types of the .NET Framework. For example, instead of Integer or int, you could use `System.Int32` in any .NET language; likewise, instead of Boolean or bool, you could use `System.Boolean`, and so on.

To sum up, once you've declared a variable as a given type, it can only hold data of that type: you can't put a string into an integer variable, for instance. However, there are frequently times when you'll need to convert one data type to another. Have a look at this code:

Visual Basic

```
Dim intX As Integer
Dim strY As String = "35"
intX = strY + 6
```

C#

```
int intX;
string strY = "35";
intX = strY + 6;
```

Now, you or I might think that this could make sense—after all, the string `strY` contains a number, so we might wish to add it to another number. Well, this isn't so simple for a computer!

VB performs some conversions for us. The VB version of the code will execute without a hitch, because the string will be converted to a number before the mathematical operation is applied. C#, on the other hand, will throw an error, as it's more strict than VB about conversions.

As a rule of thumb, it's better to stay on the safe side and avoid mixing types wherever possible.

VB and C#—Strongly Typed Languages

Even though their behavior is a little bit different, both VB and C# are **strongly typed** languages. Strongly typed languages are those that are very strict about data types. Many other languages—mostly scripting languages such as JavaScript—are loosely typed, which means that they're more flexible when it comes to dealing with data types. For example, if you try to sum a number with a string, as we did in the previous code snippet, the JavaScript interpreter would make the conversion for you automatically. At times, despite being a strongly typed language at heart, VB does a bit of background work for you, which makes it slightly easier to work with.

In .NET, you can (and sometimes need to) explicitly convert, or **cast**, the string into an integer before you're able to sum them up:

Visual Basic

```
Dim intX As Integer
Dim strY As String = "35"
intX = Int32.Parse(strY) + 6
```

C#

```
int intX;
string strY = "35";
intX = Convert.ToInt32(strY) + 6;
```

Now, the computer will accept even with the C# code, because it ends up adding two numbers, rather than a number and a string, as we tried initially. This principle holds true whenever we're mixing types in a single expression.

Arrays

Arrays are a special variety of variable that's tailored for storing related items of the same data type. Any one item in an array can be accessed using the array's name, followed by that item's position in the array (its offset). Let's create a sample page to see how it's done. The results of this code are shown in Figure 3.3:

Visual Basic File: **Arrays.aspx**

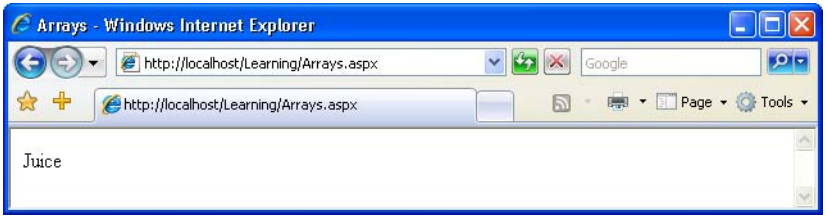
```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Arrays</title>
    <script runat="server" language="VB">
      Sub Page_Load()
        ' Declare an array
        Dim drinkList(4) As String
        ' Place some items in it
        drinkList(0) = "Water"
        drinkList(1) = "Juice"
        drinkList(2) = "Soda"
        drinkList(3) = "Milk"
        ' Access an item in the array by its position
        drinkLabel.Text = drinkList(1)
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="drinkLabel" runat="server" />
```

```
    </form>
  </body>
</html>
```

C# File: **Arrays.aspx**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Arrays</title>
    <script runat="server" language="C#">
      void Page_Load()
      {
        // Declare an array
        string[] drinkList = new string[4];
        // Place some items in it
        drinkList[0] = "Water";
        drinkList[1] = "Juice";
        drinkList[2] = "Soda";
        drinkList[3] = "Milk";
        // Access an item in the array by its position
        drinkLabel.Text = drinkList[1];
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="drinkLabel" runat="server" />
    </form>
  </body>
</html>
```

Figure 3.3. Reading an element from an array



There are some important points to pick up from this code. First, notice how we declare an array. In VB, it looks like a regular declaration for a string, except that

the number of items we want the array to contain is provided in parentheses after the name:

Visual Basic File: **Arrays.aspx (excerpt)**

```
Dim drinkList(4) As String
```

In C#, it's a little different. First, we declare that `drinkList` is an array by following the data type with two empty square brackets. We then specify that this is an array of four items, using the `new` keyword:

C# File: **Arrays.aspx (excerpt)**

```
string[] drinkList = new string[4];
```

A crucial point to realize here is that, in both C# and VB, these arrays are known as **zero-based** arrays. In a zero-based array, the first item has position `0`, the second has position `1`, and so on through to the last item, which has a position that's one less than the size of the array (`3`, in this case). So, we specify each item in our array like this:

Visual Basic File: **Arrays.aspx (excerpt)**

```
drinkList(0) = "Water"
drinkList(1) = "Juice"
drinkList(2) = "Soda"
drinkList(3) = "Milk"
```

C# File: **Arrays.aspx (excerpt)**

```
drinkList[0] = "Water";
drinkList[1] = "Juice";
drinkList[2] = "Soda";
drinkList[3] = "Milk";
```

Note that C# uses square brackets for arrays, while VB uses standard parentheses. We have to remember that arrays are zero-based when we set the label text to the second item, as shown here:

Visual Basic File: **Arrays.aspx (excerpt)**

```
drinkLabel.Text = drinkList(1)
```

C# File: **Arrays.aspx (excerpt)**

```
drinkLabel.Text = drinkList[1];
```

To help this fact sink in, you might like to try changing this code to show the third item in the list, instead of the second. Can you work out what change you'd need to make? That's right—you need only to change the number in the brackets

to reflect the new item's position in the array (don't forget to start at zero). In fact, it's this ability to select one item from a list using only its numerical location that makes arrays so useful in programming—we'll experience this first-hand as we get further into the book.

Functions

Functions are exactly the same as subroutines, but for one key difference: they return a value. In VB, we declare a function using the `Function` keyword in place of `Sub`, while in C#, we simply have to specify the return type in place of `void`. The following code shows a simple example:

Visual Basic — File: **Functions.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>ASP.NET Functions</title>
    <script runat="server" language="VB">
      ' Here's our function
      Function getName() As String
        Return "Zak Ruvalcaba"
      End Function
      ' And now we'll use it in the Page_Load handler
      Sub Page_Load(s As Object, e As EventArgs)
        messageLabel.Text = getName()
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

C# — File: **Functions.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>ASP.NET Functions</title>
    <script runat="server" language="C#">
      // Here's our function
```

```
      string getName()
      {
        return "Zak Ruvalcaba";
      }
      // And now we'll use it in the Page_Load handler
      void Page_Load()
      {
        messageLabel.Text = getName();
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

When the page above is loaded in the browser, the `Load` event will be raised which will cause the `Page_Load` event handler to be called, which in turn will call the `getName` function. Figure 3.4 shows the result in the browser.

Figure 3.4. Executing an ASP.NET function

Here's what's happening: the line in our `Page_Load` subroutine calls our function, which returns a simple string that we can assign to our label. In this simple example, we're merely returning a fixed string, but the function could just as easily retrieve the name from a database (or somewhere else). The point is that, regardless of how the function gets its data, we call it in just the same way.

When we're declaring our function, we must remember to specify the correct return type. Take a look at the following code:

Visual Basic

```
' Here's our function
Function addUp(x As Integer, y As Integer) As Integer
  Return x + y
```

```
End Function
' And now we use it in Page_Load
Sub Page_Load(s As Object, e As EventArgs)
  messageLabel.Text = addUp(5, 2).ToString()
End Sub
```

C#

```
// Here's our function
int addUp(int x, int y)
{
  return x + y;
}
// And now we use it in Page_Load
void Page_Load()
{
  messageLabel.Text = addUp(5, 2).ToString();
}
```

You can easily adapt the previous example to use this new code so that you can see the results in your browser—just replace the code inside the `<script>` tags in `Functions.aspx` with the code above.

The first thing to notice in comparing this new code to the original version of `Functions.aspx` is that our function now accepts parameters. Any function or subroutine can take any number of parameters, of any type (there's no need for parameter types to match the return type—that's just coincidental in this example).

We can readily use the parameters inside the function or subroutine just by using the names we gave them in the function declaration (here, we've chosen `x` and `y`, but we could have chosen any names).

The other difference between this and the function declaration we had before is that we now declare our function with a return type of `Integer` or `int`, rather than `String`, because we want it to return a whole number.

When we call the new function, we simply have to specify the required number of parameters, and remember that the function will return a value with the type we specify. In this case, we have to convert the integer value that the function returns to a string, so that we can assign it to the label.

The simplest way to convert an integer to a string is to append `.ToString()` to the end of the variable name. In this case, we appended `ToString` on the function

call which will return an integer during execution. Converting numbers to strings is a very common task in ASP.NET, so it's good to get a handle on it early.

Converting Numbers to Strings

There are more ways to convert numbers to strings in .NET, as the following lines of VB code illustrate:

```
messageLabel.Text = addUp(5, 2).ToString()
messageLabel.Text = Convert.ToString(addUp(5, 2))
```

If you prefer C#, these lines of code perform the same operations as the VB code above:

```
messageLabel.Text = addUp(5, 2).ToString();
messageLabel.Text = Convert.ToString(addUp(5, 2));
```

Don't be concerned if you're a little confused by how these conversions work, though—the syntax will become clear once we discuss object oriented concepts later in this chapter.

Operators

Throwing around values with variables and functions isn't of much use unless you can use them in some meaningful way, and to do so, we need operators. An **operator** is a symbol that has a certain meaning when it's applied to a value. Don't worry—operators are nowhere near as scary as they sound! In fact, in the last example, where our function added two numbers, we were using an operator: the addition operator, or + symbol. Most of the other operators are just as well known, although there are one or two that will probably be new to you. Table 3.2 outlines the operators that you'll use most often in your ASP.NET development.

Operators Abound!

The list of operators in Table 3.2 is far from complete. You can find detailed (though poorly written) lists of the differences between VB and C# operators on the Code Project web site.[3]

[3] http://www.codeproject.com/dotnet/vbnet_c__difference.asp

Table 3.2. Common ASP.NET operators

VB	C#	Description
>	>	greater than
>=	>=	greater than or equal to
<	<	less than
<=	<=	less than or equal to
<>	!=	not equal to
==	=	equals
=	=	assigns a value to a variable
OrElse	||	or
AndAlso	&&	and
&	+	concatenate strings
New	new	create object or array
*	*	multiply
/	/	divide
+	+	add
-	-	subtract

The following code uses some of these operators:

Visual Basic

```
If (user = "Zak" AndAlso itemsBought  <>  0) Then
  messageLabel.Text = "Hello Zak! Do you want to proceed to " & _
    "checkout?"
End If
```

C#

```
if (user  ==  "Zak" && itemsBought  !=  0)
{
  messageLabel.Text = "Hello Zak! Do you want to proceed to " +
    "checkout?";
}
```

Here, we use the equality, inequality (not equal to), and logical "and" operators in an `If` statement to print a tailored message for a given user when he has put a product in his electronic shopping cart. Of particular note is the C# equality operator, ==, which is used to compare two values to see if they're equal. Don't

use a single equals sign in C# unless you're assigning a value to a variable; otherwise your code will have a very different meaning than you expect!

Breaking Long Lines of Code

Since the message string in the above example was too long to fit on one line in this book, we used the string concatenation operator to combine two shorter strings on separate lines to form the complete message. In VB, we also had to break one line of code into two using the line continuation symbol (_, an underscore at the end of the line to be continued). Since C# marks the end of each command with a semicolon (`;`), you can split a single command over two lines in this language without having to do anything special.

We'll use these techniques throughout this book to present long lines of code within a limited page width. Feel free to recombine the lines in your own code if you like—there's no length limit on lines of VB and C# code.

Conditional Logic

As you develop ASP.NET applications, there will be many instances in which you'll need to perform an action only if a certain condition is met, for instance, if the user has checked a certain checkbox, selected a certain item from a `DropDownList` control, or typed a certain string into a `TextBox` control. We check for such occurrences using conditionals, the simplest of which is probably the `If` statement. This statement is often used in conjunction with an `Else` statement, which specifies what should happen if the condition is not met. So, for instance, we may wish to check whether or not the name entered in a text box is `Zak`, redirecting the user to a welcome page if it is, or to an error page if it's not:

Visual Basic

```
If (userName.Text = "Zak") Then
  Response.Redirect("ZaksPage.aspx")
Else
  Response.Redirect("ErrorPage.aspx")
End If
```

C#

```
if (userName.Text == "Zak")
{
  Response.Redirect("ZaksPage.aspx");
}
else
{
```

```
  Response.Redirect("ErrorPage.aspx");
}
```

Take Care with Case Sensitivity

Instructions are case-sensitive in both C# and VB, so be sure to use `if` in C# code, and `If` in VB code. On the other hand, variable and function names are case-sensitive only in C#. As such, in C# you could have two variables called `x` and `X`, which would be considered to be different; in VB, they would be considered to be the same variable.

Often, we want to check for many possibilities, and specify our application to perform a particular action in each case. To achieve this, we use the `Select Case` (VB) or `switch` (C#) construct:

Visual Basic

```
Select Case userName
  Case "Zak"
    Response.Redirect("ZaksPage.aspx")
  Case "Mark"
    Response.Redirect("MarksPage.aspx")
  Case "Fred"
    Response.Redirect("FredsPage.aspx")
  Case Else
    Response.Redirect("ErrorPage.aspx")
End Select
```

C#

```
switch (userName)
{
  case "Zak":
    Response.Redirect("ZaksPage.aspx");
    break;
  case "Mark":
    Response.Redirect("MarksPage.aspx");
    break;
  case "Fred":
    Response.Redirect("FredsPage.aspx");
    break;
  default:
    Response.Redirect("ErrorPage.aspx");
    break;
}
```

Loops

As you've just seen, an `If` statement causes a code block to execute once if the value of its test expression is true. Loops, on the other hand, cause a code block to execute repeatedly for as long as the test expression remains true. There are two basic kinds of loop:

- `While` loops, also called `Do` loops (which sounds like something Betty Boop might say!)
- `For` loops, including `For Next` and `For Each`

A `While` loop is the simplest form of loop; it makes a block of code repeat for as long as a particular condition is true. Here's an example:

Visual Basic File: **Loops.aspx**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Loops</title>
    <script runat="server" language="VB">
      Sub Page_Load(s As Object, e As EventArgs)
        ' Initialize counter
        Dim counter As Integer = 0
        ' Loop
        Do While counter <= 10
          ' Update the label
          messageLabel.Text = counter.ToString()
          ' We use the += operator to increase our variable by 1
          counter += 1
        Loop
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

C# File: **Loops.aspx**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Loops</title>
    <script runat="server" language="C#">
      void Page_Load()
      {
        // initialize counter
        int counter = 0;
        // loop
        while (counter <= 10)
        {
          // Update the label
          messageLabel.Text = counter.ToString();
          // C# has the ++ operator to increase a variable by 1
          counter++;
        }
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Label id="messageLabel" runat="server"/>
    </form>
  </body>
</html>
```

If you load this page, you'll get the result illustrated in Figure 3.5.

Figure 3.5. Results of a `While` loop

When you open the page, the label will be set to show the number 0, then 1, then 2, all the way to 10. Of course, since all this happens in `Page_Load` (i.e. before any output is sent to the browser), you'll only see the last value assigned: 10.

This demonstrates that the loop repeats until the condition is no longer met. Try changing the code so that the counter variable is initialized to `20` instead of `0`. When you open the page now, you won't see anything on the screen, because the loop condition was never met.

The other form of the `While` loop, called a `Do While` loop, checks whether or not the condition has been met at the end of the code block, rather than at the beginning:

Visual Basic File: **Loops.aspx (excerpt)**

```
Sub Page_Load(s As Object, e As EventArgs)
  ' Initialize counter
  Dim counter As Integer = 0
  ' Loop
  Do
    ' Update the label
    messageLabel.Text = counter.ToString()
    ' We use the += operator to increase our variable by 1
    counter += 1
  Loop While counter <= 10
End Sub
```

C# File: **Loops.aspx (excerpt)**

```
void Page_Load()
{
  // initialize counter
  int counter = 0;
  // loop
  do
  {
    // Update the label
    messageLabel.Text = counter.ToString();
    // C# has the operator ++ to increase a variable by 1
    counter++;
  }
  while (counter <= 10);
}
```

If you run this code, you'll see it provides the exact same output we saw when we tested the condition before the code block. However, we can see the crucial difference if we change the code so that the counter variable is initialized to `20`. In this case, we will, in fact, see 20 displayed, because the loop code is executed once before the condition is even checked! There are some instances when this

is just what we want, so being able to place the condition at the end of the loop can be very handy.

A `For` loop is similar to a `While` loop, but we typically use it when we know beforehand how many times we need it to execute. The following example displays the count of items within a `DropDownList` control called `productList`:

Visual Basic

```
Dim i As Integer
For i = 1 To productList.Items.Count
  messageLabel.Text = i.ToString()
Next
```

C#

```
int i;
for (i = 1; i <= productList.Items.Count; i++)
{
  messageLabel.Text = i.ToString();
}
```

In VB, the loop syntax specifies the starting and ending values for our counter variable within the `For` statement itself.

In C#, we assign a starting value (`i = 1`) along with a condition that will be tested each time we move through the loop (`i <= productList.Items.Count`), and identify how the counter variable should be incremented after each loop (`i++`). While this allows for some powerful variations on the theme in our C# code, it can be confusing at first. In VB, the syntax is considerably simpler, but it can be a bit limiting in exceptional cases.

The other type of `For` loop is `For Each`, which loops through every item within a collection. The following example loops through an array called `arrayName`:

Visual Basic

```
For Each item In arrayName
  messageLabel.Text = item
Next
```

C#

```
foreach (string item in arrayName)
{
  messageLabel.Text = item;
}
```

You may also come across instances in which you need to exit a loop prematurely. In these cases, you would use either `Exit`, if your code is in VB, or the equivalent (`break`) statement in C#, to terminate the loop:

Visual Basic

```
Dim i As Integer
For i = 0 To 10
  If (i = 5) Then
    Response.Write("Oh no! Not the number 5!!")
    Exit For
  End If
Next
```

C#

```
int i;
for (i = 0; i <= 10; i++)
{
  if (i == 5)
  {
    Response.Write("Oh no! Not the number 5!!");
    break;
  }
}
```

In this case, as soon as our `For` loop hits the condition `i = 5`, it displays a warning message using the `Response.Write` method (which will be familiar to those with past ASP experience), and exits the loop so that no further passes through the loop will be made.

Although we've only scratched the surface, VB and C# provide a great deal of power and flexibility to web developers, and time spent learning the basics now will more than pay off in the future.

Object Oriented Programming Concepts

VB and C# are modern programming languages that give you the tools to write structured, extensible, and maintainable code. The code can be separated into modules, each of which defines classes that can be imported and used in other modules. Both languages are relatively simple to get started with, yet they offer sophisticated features for writing complex, large-scale enterprise applications.

One of the reasons why these languages are so powerful is that they facilitate **object oriented programming** (OOP). In this section, we'll explain the funda-

mentals of OOP and learn how adopting good OOP style now can help you to develop better, more versatile web applications down the road. This section will provide a basic OOP foundation angled towards the web developer. In particular, we'll cover the following concepts:

- objects
- properties
- methods
- classes
- scope
- events
- inheritance

In the pages that follow, we'll discuss these concepts briefly, and from Chapter 4 onwards, you'll see some practical examples of OOP in action.

Objects and Classes

So what does the term "object oriented programming" mean? Basically, as the name suggests, it's an approach to development that puts objects at the center of the programming model. The object is probably the most important concept in the world of OOP; an **object** is a self-contained entity that has *state* and *behavior*, just like a real-world object.

In programming, an object's state is described by its **fields** and **properties**, while its behavior is defined by its **methods** and **events**. An important part of OOP's strength comes from the natural way it allows programmers to conceive and design their applications.

We often use objects in our programs to describe real-world objects—we can have objects that represent a car, a customer, a document, or a person. Each object has its own state and behavior.

It's very important to have a clear understanding of the difference between a class and an object. A class acts like a blueprint for the object, while an object represents an instance of the class. I just said that you could have objects of type

`Car`, for example. If you did, `Car` would be the class, or the type, and we could create as many `Car` objects as we wanted, calling them `myCar`, `johnsCar`, `davesCar`, and so on.

The class defines the behavior of all objects of that type. So all objects of type `Car` will have the same *behavior*—for example, the ability to change gear. However, each individual `Car` object may be in a different gear at any particular time; thus, each object has its own particular *state*.

Let's take another example: think of `Integer` (or `int`) as a class, and `age` and `height` as objects of type `Integer`. The class defines the behavior of the objects—they're numeric, and we can perform mathematical operations on them—and the instances of objects (`age` and `height`) have their behavior defined by the class to which they belong, but they also hold state (so `age` could be `20`).

Take a look at the following code:

Visual Basic

```
Dim age As Integer
Dim name As String
Dim myCar as Car
Dim myOtherCar as Car
```

C#

```
int age;
string name;
Car myCar;
Car myOtherCar;
```

As you can see, the syntax for declaring an object is the same as that for declaring a simple integer or string variable. In C#, we first mention the type of the object, then we name that particular instance. In VB, we use the `Dim` keyword.

Object oriented programming sounds like an advanced topic, but getting started with it is actually very easy, because OOP offers us a natural way to conceive and design programs. Instead of writing long functions of code to perform specific tasks, OOP allows us to group pieces of related functionality into classes that we can reuse over and over, or even extend to incorporate new features. In OOP, one thinks of programming problems in terms of objects, properties, and methods. And, as we've seen, the best way to get a handle on these terms is to consider a real-world object and imagine how it might be represented in an OOP program. For the examples that follow, we'll use as our example my dog, an Australian Shepherd named Rayne.

Rayne is your average great big, friendly, loving, playful mutt. You might describe him in terms of his physical properties: he's gray, white, brown, and black, stands roughly one-and-a-half feet high, and is about three feet long. You might also describe some methods to make him do things: he sits when he hears the command "Sit," lies down when he hears the command "Lie down," and comes when his name is called.

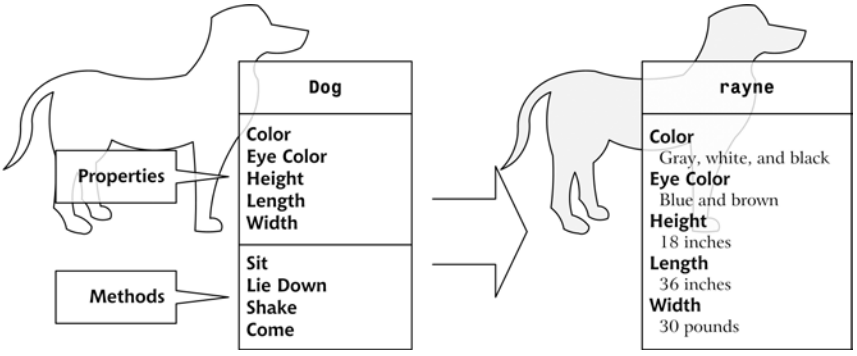
So, if we were to represent Rayne in an OOP program, we'd probably start by creating a class called `Dog`. A class describes how certain types of objects look from a programming point of view. When we define a class, we must define the following two items:

Properties Properties hold specific information relevant to that class of object. You can think of properties as characteristics of the objects that they represent. Our `Dog` class might have properties such as `Color`, `Height`, and `Length`.

Methods Methods are actions that objects of the class can be told to perform. Methods are subroutines (if they don't return a value) or functions (if they do) that are specific to a given class. So the `Dog` class could have methods such as `Sit`, and `LieDown`.

Once we've defined a class, we can write code that creates objects of that class, using the class a little like a template. This means that objects of a particular class expose (or make available) the methods and properties defined by that class. So, we might create an instance of our `Dog` class called `rayne`, set its properties accordingly, and use the methods defined by the class to interact with Rayne, as shown in Figure 3.6.

Figure 3.6. An instance of `Dog`



This is just a simple example to help you visualize what OOP is all about. In the next few sections, we'll cover properties and methods in greater detail, and talk about classes and class instances, scope, events, and inheritance.

Properties

As we've seen, properties are characteristics shared by all objects of a particular class. In the case of our example, the following properties might be used to describe any given dog:

- color
- height
- length

In the same way, the more useful ASP.NET `Button` class exposes properties including:

- `Width`
- `Height`
- `ID`
- `Text`
- `ForeColor`
- `BackColor`

Unfortunately for me, if I get sick of Rayne's color, I can't change it in real life. However, if Rayne was a .NET object, we could change any of his properties in the same way that we set variables (although a property can be read-only or write-only). For instance, we could make him brown very easily:

Visual Basic

```
rayne.Color = "Brown"
```

C#

```
rayne.Color = "Brown";
```

In this example, we're using an instance of our `Dog` class called `rayne`. We use the dot operator (`.`) to access the property `Color` that the object exposes and set it to the string "`Brown`."

Methods

With our dog example, we can make a particular dog do things by calling commands. If I want Rayne to sit, I tell him to sit. If I want Rayne to lie down, I tell him to lie down. In object oriented terms, I tell him what I want him to do by calling a predefined command or method, and an action results. For example, to make Rayne sit, we would use the following code to call his `Sit` method:

Visual Basic

```
rayne.Sit()
```

C#

```
rayne.Sit();
```

Given that `rayne` is an instance of our `Dog` class, we say that the `Sit` method is exposed by the `Dog` class.

Classes

You can think of a class as a template for building as many objects of a particular type as you like. When you create an instance of a class, you are creating an object of that class, and the new object has all the characteristics and behaviors (properties and methods) defined by the class.

In our dog example, `rayne` was an instance of the `Dog` class, as Figure 3.6 illustrated. In our code, we'd create a new instance of the `Dog` class called `rayne`, as shown below:

Visual Basic

```
Dim rayne As New Dog()
```

C#

```
Dog rayne = new Dog();
```

Constructors

Constructors are special kinds of method are that used to initialize the object. In OOP, when we create new instances of a class, we say we're **instantiating**

that class. The constructor is a method of a class that's executed automatically when a class is instantiated.

At least one constructor will be defined for most of the classes you will write (though we can define more than one constructor for a class, as we'll see shortly), since it's likely that some data will need to be initialized for each class at creation time.

In C# and VB, the constructor is defined as a method that has the same name as the class, and has no return type.

Scope

You should now understand programming objects to be entities that exist in a program and are manipulated through the methods and properties they expose. However, in some cases, we want to create for use inside our class methods that are not available to code outside that class.

Imagine we're writing the `Sit` method inside this class, and we realize that before the dog can sit, it has to shuffle its back paws forward a little (bear with me on this one!). We could create a method called `ShufflePaws`, then call that method from inside the `Sit` method. However, we don't want code in an ASP.NET page or in some other class to call this method—it'd just be silly. We can prevent it by controlling the scope of the `ShufflePaws` method.

Carefully controlling which members of a class are accessible from outside that class is fundamental to the success of object oriented programming. You can control the visibility of a class member using a special set of keywords called **access modifiers**:

`Public` Defining a property or method of a class as public allows that property or method to be called from outside the class itself. In other words, if an instance of this class is created inside another object (remember, too, that ASP.NET pages themselves are objects), public methods and properties are freely available to the code that created that instance of the class. This is the default scope for VB and C# classes.

`Private` If a property or method of a class is private, it cannot be used from outside the class itself. So, if an instance of this class is created inside an object of a different class, the creating object has no access to private methods or properties of the created object.

`Protected` A protected property or method sits somewhere between public and private. A protected member is accessible from the code within its class, or to the classes derived from it. We'll learn more about derived classes a bit later.

Deciding which access modifier to use for a given class member can be a very difficult decision—it affects not only your class, but also the other classes and programs that use your class. Of special importance are the class's public members, which together form the class's **public interface**. The public interface acts like a contract between your class and the users of your class, and if it's designed properly, it shouldn't change over time. If, for example, you mark the `Sit` method as public, and later decide to make it private, all the other classes that use this method will have to change accordingly, which is not good. For an extreme scenario, imagine that in a year, Microsoft decided to remove the `ToString` method from its classes—obviously, this would wreak havoc with your code.

Keep Everything Private until you Need It

As a simple guideline for designing your classes, remember that it's often easier just to make all the members private, and make public only those that really need to be public. It's much easier to add to a public interface than it is to remove from it.

Events

We've covered events in some depth already. To sum up, events occur when a control object sends a message as a result of some change that has been made to it. Generally, these changes occur as the result of user interaction with the control via the browser. For instance, when a button is clicked, a `Click` event is raised, and we can handle that event to perform some action. The object that triggers the event is referred to as the **event sender**, while the object that receives the event is referred to as the **event receiver**. You'll learn more about these objects in Chapter 4.

Understanding Inheritance

The term **inheritance** refers to the ability of a specialized class to refine the properties and methods exposed by another, more generalized class.

In our dog example, we created a class called `Dog`, then created instances of that class to represent individual dogs such as Rayne. However, dogs are types of animals, and many characteristics of dogs are shared by all (or most) animals. For

instance, Rayne has four legs, two ears, one nose, two eyes, etc. It might be better, then, for us to create a base class called `Animal`. When we then defined the `Dog` class, it would inherit from the `Animal` class, and all public properties and methods of `Animal` would be available to instances of the `Dog` class.

Similarly, we could create a new class based on the `Dog` class. In programming circles, this is called **deriving a subclass** from `Dog`. For instance, we might create a class called `AustralianShepherd`, and one for my other dog, Amigo, called `Chihuahua`, both of which would inherit the properties and methods of the `Dog` base class, and define new classes specific to each breed.

Don't worry too much if this is still a little unclear. The best way to appreciate inheritance is to see it used in a real program. The most obvious use of inheritance in ASP.NET is in the technique called code-behind, and we'll build plenty of examples using inheritance and code-behind in Chapter 4.

Objects In .NET

If this is the first book in which you've read about object oriented programming, you've probably started to dream about objects! Don't worry, the effect of first exposure to objects doesn't usually last for more than a week. Even though this is yet another discussion about objects, I promise it won't be boring. Moreover, in the course of this section, we'll cover some important concepts that every serious .NET programmer must know.

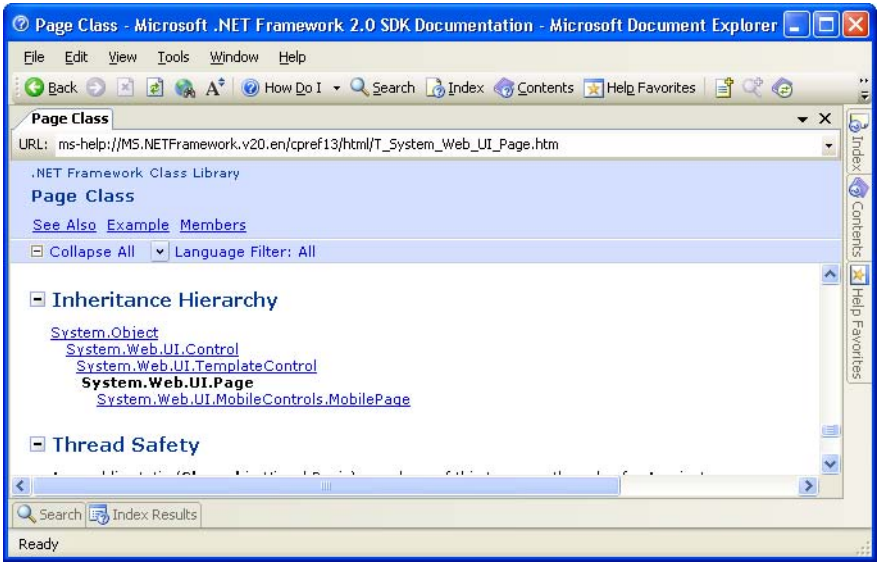
So far, we've explored various concepts that apply in one form or the other to almost any truly object oriented language. Every language has its peculiarities, but the general concepts are the same in all of these languages.

You may already have heard the common mantra of object oriented programmers: "everything is an object." This has two meanings. First of all, in C#, every program consists of a class. In all stages of application development, from design to implementation, decisions must be made in regard to the way we design and relate objects and classes to each other. Yes, objects are everywhere.

.NET extends this to yet another level, giving the phrase "everything is an object" extra meaning. In the world of .NET, every class ultimately derives from a base class named `Object`, so "everything is an object" becomes "everything is an `Object`."

If you look at the documentation for the ASP.NET `Page` class, you can see the list of classes from which this class inherits, as shown in Figure 3.7.

Figure 3.7. The `Page` class's documentation



You'll remember from the last section that we said our hypothetical `AustralianShepherd` class would inherit from the more general `Dog` class, which, in turn, would inherit from the even more general `Animal` class. This is exactly the kind of relationship that's being shown in Figure 3.7—`Page` inherits methods and properties from the `TemplateControl` class, which in turn inherits from a more general class called `Control`. In the same way that we say that an Australian Shepherd is an Animal, we say that a `Page` is a `Control`. `Control`, like all .NET classes, inherits from `Object`.

Since `Object` is so important that every other class derives from it, either directly or indirectly, it deserves a closer look. `Object` contains the basic functionality that the designers of .NET felt should be available in any object. The `Object` class contains these public members:

- `Equals`
- `ReferenceEquals`
- `GetHashCode`
- `GetType`

- `ToString`

The only member we're really interested in at this moment is `ToString`, which returns the text representation of an object. This method is called automatically when conversions to string are needed, as is the case in the following code, which joins a number and a string:

Visual Basic

```
Dim age As Integer = 5
Dim message As String = "Current Age: " & age
```

C#

```
int age = 5;
string message = "Current Age: " + age;
```

Namespaces

As ASP.NET is part of the .NET Framework, we have access to all the goodies that are built into it in the form of the .NET Framework Class Library. This library represents a huge resource of tools and features in the form of classes; these classes are organized in a hierarchy of namespaces. When we want to use certain features that .NET provides, we have only to find the namespace that contains the desired functionality, and import that namespace into our ASP.NET page. Once we've done that, we can make use of the .NET classes in that namespace to achieve our own ends.

For instance, if we wanted to access a database from a page, we would import the namespace that contains classes for this purpose, which could be `System.Data.SqlClient`. You can view the namespace of a class when visiting its page in the .NET documentation. For example, the `Button` control's class can be found in `System.Web.UI.WebControls`.

To use a class that's part of a namespace that isn't available to you by default, you either need to import the namespace, or reference the class using its **fully qualified name**, such as `System.Web.UI.WebControls`. To import a namespace page, we use the `Imports` directive in VB, and `using` in C#:

Visual Basic

```
Imports System.Data.SqlClient
```

C#

```
using System.Data.SqlClient;
```

As we've imported that namespace, we have access to all the classes that it contains.

Using Code-behind Files

Most companies that employ web development teams usually split projects into two groups—visual design and functional development—because software engineers are usually poor designers, and designers are often poor engineers. Until now, our ASP.NET pages have contained code render blocks that place VB or C# code directly into the ASP.NET page. The problem with this approach is that there's no separation between the presentational elements of the page and the application logic. Traditional ASP was infamous for creating "spaghetti" code, which was scattered and intertwined throughout the presentation elements. This made it very tricky to manage the code between development teams, as you'll know if you've ever tried to pick apart someone else's ASP code. In response to these problems, ASP.NET introduced a new development approach that allows code developers to work separately from the presentation designers who lay out individual pages.

This new method, called **code-behind**, keeps all of your presentational elements (controls) inside the `.aspx` file, but moves all of your code to a separate class in a `.vb` or `.cs` code-behind file. Consider the following ASP.NET page, which displays a simple button and label:

Visual Basic — File: **`Hello.aspx (excerpt)`**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page using VB</title>
    <script runat="server" language="VB">
      Sub Click(s As Object, e As EventArgs)
        messageLabel.Text = "Hello World"
      End Sub
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Button id="submitButton" Text="Click Me"
          runat="server" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
```

```
  </body>
</html>
```

C# File: **Hello.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page using C#</title>
    <script runat="server" language="C#">
      void Click(Object s, EventArgs e)
      {
        messageLabel.Text = "Hello World";
      }
    </script>
  </head>
  <body>
    <form runat="server">
      <asp:Button id="submitButton" Text="Click Me"
           runat="server" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

Let's see how this example could be separated into the following distinct files:

`HelloCodeBehind.aspx`
: layout, presentation, and static content

`HelloCodeBehind.vb` or **`HelloCodeBehind.cs`**
: code-behind files containing a custom page class

First, we take all the code and place it in the code-behind file (`HelloCodeBehind.vb` or `HelloCodeBehind.cs`). This file is a pure code file, and contains no HTML or other markup tags. Nevertheless, we can still access presentation elements from this file, using their IDs (such as `messageLabel`) as shown below:

Visual Basic File: **HelloCodeBehind.vb**

```
' First off we import some useful namespaces
Imports System
Imports System.Web.UI
Imports System.Web.UI.WebControls
' The partial class
Public Partial Class HelloCodeBehind
```

```
  Inherits System.Web.UI.Page
  ' Here's the Click handler just as it appeared before
  Sub Click(s As Object, e As EventArgs)
    messageLabel.Text = "Hello World"
  End Sub
End Class
```

C# File: **HelloCodeBehind.cs**

```
// First off we import some useful namespaces
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
// The partial class
public partial class HelloCodeBehind: System.Web.UI.Page
{
  // Here's the Click handler just as it appeared before
  public void Click(Object s, EventArgs e)
  {
    messageLabel.Text = "Hello World";
  }
}
```

Without code, the main ASP.NET page becomes a bit simpler:

Visual Basic File: **HelloCodeBehind.aspx**

```
<%@ Page Language="VB" CodeFile="HelloCodeBehind.vb"
    Inherits="HelloCodeBehind"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page Code Behind Demo using VB</title>
  </head>
  <body>
    <form runat="server">
      <asp:Button id="submitButton" Text="Click Me"
          runat="server" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

C# File: **HelloCodeBehind.aspx**

```
<%@ Page Language="C#" CodeFile="HelloCodeBehind.cs"
    Inherits="HelloCodeBehind"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
```

```
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Sample Page Code Behind Demo using C#</title>
  </head>
  <body>
    <form runat="server">
      <asp:Button id="submitButton" Text="Click Me"
          runat="server" OnClick="Click" />
      <asp:Label id="messageLabel" runat="server" />
    </form>
  </body>
</html>
```

As you can see, the only line that differs between these `.aspx` pages is the `Page` directive. Since the `.aspx` pages now contain only HTML layout, the contents are identical no matter what language you use for the code.

Partial Classes

If you have programmed with ASP.NET 1.1, you may already have noticed the changes in the code-behind model. In ASP.NET 2.0, the code-behind file is cleaner and smaller—a feat it achieves by using a new feature of VB and C# called **partial classes**. Read on for the details!

The code-behind file is written differently from what you've seen so far. While we no longer need `<script>` tags, we find a class definition in their place. As the VB example shows, we start with three lines that import namespaces for use within the code:

Visual Basic — File: **HelloCodeBehind.vb (excerpt)**

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

The next lines create a new class, named `HelloCodeBehind`. Since our code-behind page contains code for an ASP.NET page, our class inherits from the `Page` class:

Visual Basic — File: **HelloCodeBehind.vb (excerpt)**

```
Public Partial Class HelloCodeBehindSample
  Inherits System.Web.UI.Page
```

This is the practical application of inheritance that we mentioned earlier. The `HelloCodeBehind` class inherits from `Page`, borrowing all its functionality, and extending it according to the particular needs of the page.

But what does `Partial` mean? A new feature in .NET 2.0, partial classes allow a class to be spread over multiple files. ASP.NET 2.0 uses this feature to make programmers' lives easier. We write one part of the class in the code-behind file, and ASP.NET generates the other part of the class for us, adding the object declarations for all the user interface elements.

Take a look at the `Click` subroutine, though which we access the `messageLabel` object without defining it anywhere in the code:

Visual Basic File: **`HelloCodeBehind.vb (excerpt)`**

```
Sub Click(s As Object, e As EventArgs)
  messageLabel.Text = "Hello World"
End Sub
```

That's pretty handy! However, don't be fooled into thinking that you can use objects that haven't been declared—the `messageLabel` object has been declared in another partial class file that the ASP.NET runtime generates for us. The file contains declarations for all the controls referenced in `HelloCodeBehind.aspx`.

As I hope you can see, code-behind files are easy to work with, and they can make managing and using our pages much more straightforward than keeping your code in code declaration blocks. You'll find yourself using code-behind files in most of the real-world projects that you build, but for simplicity's sake, we'll stick with code declaration blocks for one more chapter.

Summary

Phew! We've covered quite a few concepts over the course of this chapter. Don't worry—with a little practice, these concepts will become second nature to you. I hope you leave this chapter with a basic understanding of programming concepts as they relate to the ASP.NET web developer.

The next chapter will begin to put all the concepts that we've covered so far into practice. We'll begin by working with HTML Controls, Web Forms, and Web Controls, before launching into our first hands-on project!

4

Constructing ASP.NET Web Pages

If you've ever built a model from Lego bricks, you're well prepared to start building real ASP.NET web pages. ASP.NET offers many techniques that allow web developers to build parts of web pages independently, then put them together later to build complete pages.

The content we're organizing through our work with ASP.NET is almost never static. At design time, we tend to think in terms of templates that contain placeholders for the content that will be generated dynamically at runtime. And to fill those placeholders, we can either use one of the many controls ASP.NET provides, or build our own.

In this chapter, we'll discuss many of the objects and techniques that give life and color to ASP.NET web pages, including:

- web forms
- HTML server controls
- web server controls
- web user controls
- master pages

- handling page navigation
- styling pages and controls with CSS

If the list looks intimidating, don't worry—all of this is far easier to understand than it might first appear.

Web Forms

As you know, there's always new terminology to master when you're learning new technologies. But with ASP.NET, even the simplest terms that are used to describe the basics of web pages change to reflect the processes that occur within them.

The term used to describe an ASP.NET web page is **web form**, and this is the central object in ASP.NET development. You've already met web forms—they're the `.aspx` files you've worked with so far in this book. At first glance, web forms look much like HTML pages, but in addition to static HTML content they also contain ASP.NET presentational elements, and code that executes on the server side to generate dynamic content and perform the desired server-side functionality.

Every web form includes a `<form runat="server">` tag, which contains the ASP.NET-specific elements that make up the page. Multiple `form`s aren't supported. The basic structure of a web form is shown here:

```
<html>
  <head>
    <script runat="server" language="language">
      …code here…
    </script>
  </head>
  <body>
    <form runat="server">
      …user interface elements here…
    </form>
  </body>
</html>
```

To access and manipulate a web form programatically, we use the `System.Web.UI.Page` class. You might recognize this class from the code-behind example we saw in Chapter 3. We must mention the class explicitly in the code-behind file. In situations in which we're not using code-behind files (i.e. we write

all the code inside the `.aspx` file instead), the `Page` class is still used—we just don't see it.

We can use a range of user interface elements inside the form—including typical, static HTML code—but we can also use elements whose values or properties can be generated or manipulated on the server either when the page first loads, or when the form is submitted. These elements—which, in ASP.NET parlance, are called controls—allow us to reuse common functionality, such as the page header, a calendar, a shopping cart summary, or a "Today's Quote" box, for example, across multiple web forms. There are several types of controls in ASP.NET:

- HTML server controls
- web server controls
- web user controls
- master pages

There are significant technical differences between these types of controls, but what makes them similar is the ease with which we can integrate and reuse them in our web sites. Let's take a look at them one by one.

HTML Server Controls

HTML server controls are outwardly identical to plain old HTML tags, but include a `runat="server"` attribute. This gives the ASP.NET runtime control over the HTML server controls, allowing us to access them programatically. For example, if we have an `<a>` tag in a page and we want to be able to change the address to which it links dynamically, using VB or C# code, we use the `runat="server"` attribute.

A server-side HTML server control exists for each of HTML's most common elements. Creating HTML server controls is easy: we simply stick a `runat="server"` attribute on the end of a normal HTML tag to create the HTML control version of that tag. The complete list of current HTML control classes and their associated tags is given in Table 4.1.

Table 4.1. HTML control classes

Class	Associated Tags
`HtmlAnchor`	`<a runat="server">`
`HtmlButton`	`<button runat="server">`
`HtmlForm`	`<form runat="server">`
`HtmlImage`	`<img runat="server">`
`HtmlInputButton`	`<input type="submit" runat="server">`
	`<input type="reset" runat="server">`
	`<input type="button" runat="server">`
`HtmlInputCheckBox`	`<input type="checkbox" runat="server">`
`HtmlInputFile`	`<input type="file" runat="server">`
`HtmlInputHidden`	`<input type="hidden" runat="server">`
`HtmlInputImage`	`<input type="image" runat="server">`
`HtmlInputRadioButton`	`<input type="radio" runat="server">`
`HtmlInputText`	`<input type="text" runat="server">`
	`<input type="password" runat="server">`
`HtmlSelect`	`<select runat="server">`
`HtmlTable`	`<table runat="server">`
`HtmlTableRow`	`<tr runat="server">`
`HtmlTableCell`	`<td runat="server">`
	`<th runat="server">`
`HtmlTextArea`	`<textarea runat="server">`
`HtmlGenericControl`	`<span runat="server">`
	`<div runat="server">`
	All other HTML tags

For more details on these classes, see Appendix A.

All the HTML server control classes are contained within the `System.Web.UI.HtmlControls` namespace. As they're processed on the server side by the ASP.NET runtime, we can access their properties through code elsewhere in the page. If you're familiar with JavaScript, HTML, and CSS, then you'll know that manipulating text within HTML tags, or even manipulating inline styles within an HTML

tag, can be cumbersome and error-prone. HTML server controls aim to solve these problems by allowing you to manipulate the page easily with your choice of .NET language—for instance, using VB or C#.

Using the HTML Server Controls

Nothing explains the theory better than a simple, working example. Let's create a simple survey form that uses the following HTML server controls:

- `HtmlForm`
- `HtmlButton`
- `HtmlInputText`
- `HtmlSelect`

We'll begin by creating a new file named `Survey.aspx`. Create the file in the `Learning` folder you created in Chapter 1. The following code creates the visual interface for the survey:

File: **`Survey.aspx` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Using ASP.NET HTML Server Controls</title>
    <!-- code will go here -->
  </head>
  <body>
    <form runat="server">
      <h2>Take the Survey!</h2>
      <!-- Display user name -->
      <p>
        Name:<br />
        <input type="text" id="name" runat="server" />
      </p>
      <!-- Display email -->
      <p>
        Email:<br />
        <input type="text" id="email" runat="server" />
      </p>
      <!-- Display technology options -->
      <p>
```

```
        Which server technologies do you use?<br />
        <select id="serverModel" runat="server" multiple="true">
          <option>ASP.NET</option>
          <option>PHP</option>
          <option>JSP</option>
          <option>CGI</option>
          <option>ColdFusion</option>
        </select>
      </p>
      <!-- Display .NET preference options -->
      <p>
        Do you like .NET so far?<br />
        <select id="likeDotNet" runat="server">
          <option>Yes</option>
          <option>No</option>
        </select>
      </p>
      <!-- Display confirmation button -->
      <p>
        <button id="confirmButton" OnServerClick="Click"
            runat="server">Confirm</button>
      </p>
      <!-- Confirmation label -->
      <p>
        <asp:Label id="feedbackLabel" runat="server" />
      </p>
    </form>
  </body>
</html>
```

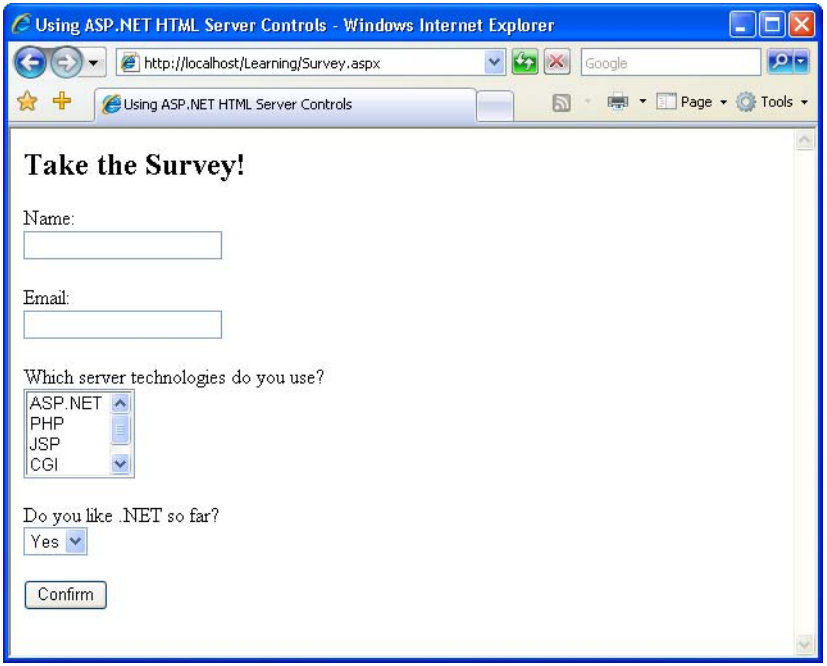
From what we've already seen of HTML controls, you should have a good idea of the classes we'll be working with in this page. All we've done is place some `HtmlInputText` controls, an `HtmlButton` control, and an `HtmlSelect` control inside the obligatory `HtmlForm` control. We've also added a `Label` control, which we'll use to give feedback to the user.

HTML Server Controls in Action

Remember, HTML server controls are essentially HTML tags with the `runat="server"` attribute. In most cases, you'll also need to assign them IDs, which will enable you to use the controls in your code.

When it's complete, the `Survey.aspx` web form will resemble Figure 4.1.

Figure 4.1. A simple form that uses HTML server controls



When a user clicks on the button, we'll display the submitted responses in the browser. In a real application, we'd probably be more likely to save this information to a database and perhaps show the results as a chart. Whatever the case, we'd access the properties of the HTML controls as shown below:

Visual Basic — File: **Survey.aspx (excerpt)**

```
<script runat="server" language="VB">
  Sub Click(ByVal s As Object, ByVal e As EventArgs)
    Dim i As Integer
    feedbackLabel.Text = "Your name is: " & name.Value & "<br />"
    feedbackLabel.Text += "Your email is: " & email.Value & _
        "<br />"
    feedbackLabel.Text += "You like to work with:<br />"
    For i = 0 To serverModel.Items.Count - 1
      If serverModel.Items(i).Selected Then
        feedbackLabel.Text += " - " & _
            serverModel.Items(i).Text & "<br />"
      End If
    Next i
```

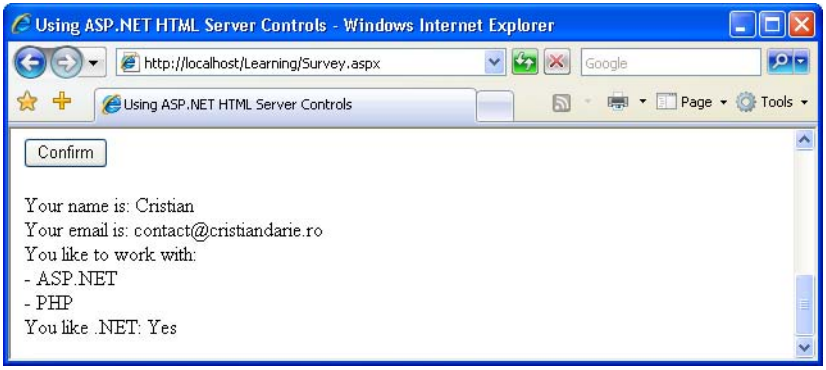
```
    feedbackLabel.Text += "You like .NET: " & likeDotNet.Value
  End Sub
</script>
```

C# File: **Survey.aspx (excerpt)**

```
<script runat="server" language="C#">
  void Click(Object s, EventArgs e)
  {
    feedbackLabel.Text = "Your name is: " + name.Value + "<br />";
    feedbackLabel.Text += "Your email is: " + email.Value +
        "<br />";
    feedbackLabel.Text += "You like to work with:<br />";
    for (int i = 0; i <= serverModel.Items.Count - 1; i++)
    {
      if (serverModel.Items[i].Selected)
      {
        feedbackLabel.Text += " - " + serverModel.Items[i].Text +
            "<br />";
      }
    }
    feedbackLabel.Text += "You like .NET: " + likeDotNet.Value;
  }
</script>
```

As with the examples in previous chapters, we start by placing our VB and C# code inside a server-side script block within the `<head>` part of the page. Next, we create a new `Click` event handler that takes the two usual parameters. Finally, we use the `Label` control to display the user's responses within the page.

Figure 4.2. Viewing the survey results



Once you've written the code, save your work and test the results in your browser. Enter some information and click the button. To select multiple options in the `serverModel` option box, hold down **Ctrl** as you click on your preferences. The information you enter should appear at the bottom of the page when the Confirm button is clicked, as shown in Figure 4.2.

In conclusion, working with HTML server controls is really simple. All you need to do is assign each control an ID, and add the `runat="server"` attribute. Then, you can simply access and manipulate them using VB or C# code on the server side.

Web Server Controls

Web server controls can be seen as more advanced versions of HTML server controls. Web server controls are those that generate content for you—you're no longer in control of the HTML being used. While having good knowledge of HTML is useful, it's not a necessity for those working with web server controls.

Let's look at an example. We can use the `Label` web server control to place simple text inside a web form. To change the `Label`'s text from within our C# or VB code, we simply set its `Text` property like so:

Visual Basic

```
myLabel.Text = "Mickey Mouse"
```

Similarly, to add a text box to our form, we use the `TextBox` web server control. Again, we can read or set its text using the `Text` property:

C#

```
username = usernameTextBox.Text;
```

Though we're applying the `TextBox` control, ASP.NET still uses an `input` element behind the scenes; however, we no longer have to worry about this detail. With web server controls, Microsoft has basically reinvented HTML from scratch.

Unlike HTML server controls, web server controls don't have a direct, one-to-one correspondence with the HTML elements they generate. For example, we can use either of two web server controls—the `DropDownList` control, or the `ListBox` control—to generate a `select` element.

Web server controls follow the same basic pattern as HTML tags, but the tag name is preceded by `asp:`, and is capitalized using Pascal Casing. Pascal Casing is a form that capitalizes the first character of each word (e.g. `TextBox`). The

object IDs are usually named using Camel Casing, where the first letter of each word except the first is capitalized (e.g. `usernameTextBox`).

Consider the following HTML `input` element, which creates an input text box:

```
<input type="text" name="usernameTextBox" size="30" />
```

The equivalent web server control is the `TextBox` control, and it looks like this:

```
<asp:TextBox id="usernameTextBox" runat="server" Columns="30">
</asp:TextBox>
```

Remember that, unlike any normal HTML that you might use in your web forms, web server controls are first processed by the ASP.NET engine, where they're transformed to HTML. A side effect of this approach is that you must be very careful to always include closing tags (the `</asp:TextBox>` part above). The HTML parsers of most web browsers are forgiving about badly formatted HTML code, but ASP.NET is not. Remember that you can use the shorthand `/>` syntax if nothing appears between your web server control's opening and closing tags. So, you could also write this `TextBox` like so:

```
<asp:TextBox id="usernameTextBox" runat="server" Columns="30" />
```

To sum up, the key points to remember when working with web server controls are:

- Web server controls must be placed within a `<form runat="server">` tag to function properly.
- Web server controls require the `runat="server"` attribute to function properly.
- We include web server controls in a form using the `asp:` prefix.

There are more web server controls than HTML controls, some offer advanced features that simply aren't available using HTML alone, and some generate quite complex HTML code for you. We'll meet many of the web server controls as we work through this and future chapters.

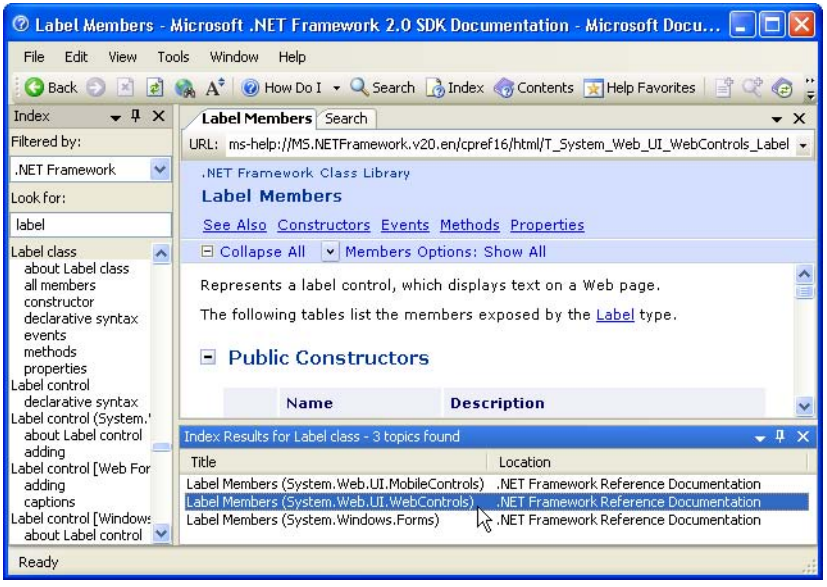
For more information on web server controls, including the properties, methods, and events for each, have a look at Appendix B.

Standard Web Server Controls

The standard set of web server controls that comes with ASP.NET mirrors the HTML server controls in many ways. However, web server controls offer some new refinements and enhancements, such as support for events and view state, a more consistent set of properties and methods, and more built-in functionality. In this section, we'll take a look as some of the controls you're most likely to use in your day-to-day work.

Remember to use the .NET Framework 2.0 SDK Documentation whenever you need more details about any of the framework's classes (or controls). Access the documentation from Start > All Programs > Microsoft .NET Framework SDK v2.0 > Documentation. To find a class, simply search for the class's name. If there are many classes with a given name in different namespaces, you'll be able to choose the one you want from the Index Results window. For example, you'll find that there are three classes named `Label`, situated in the `System.Web.UI.MobileControls`, `System.Web.UI.WebControls`, and `System.Windows.Forms` namespaces, as Figure 4.3 illustrates. You'll most likely be interested in the version of the class situated in the `WebControls` namespace.

Figure 4.3. Documentation for the `Label` control



Label

The easiest way to display static text on your page is simply to add the text to the body of the page without enclosing it in any tag. However, if you want to modify the text displayed on a page using ASP.NET code, you can display your text within a `Label` control. Here's a typical example:

```
<asp:Label id="messageLabel" Text="" runat="server" />
```

The following code sets the `Text` property of the `Label` control to display the text "Hello World":

Visual Basic

```
Public Sub Page_Load()
  messageLabel.Text = "Hello World"
End Sub
```

C#

```
public void Page_Load()
{
  messageLabel.Text = "Hello World";
}
```

Reading this `Page_Load` handler code, we can see that when the page first loads, the `Text` property of the `Label` control with the `id` of `message` will be set to "Hello World."

Literal

This is perhaps the simplest control in ASP.NET. If you set `Literal`'s `Text` property, it will simply insert that text into the output HTML code without altering it. Unlike `Label`, which has similar functionality, `Literal` doesn't wrap the text in `<span>` tags that would allow the setting of style information.

TextBox

The `TextBox` control is used to create a box in which the user can type or read standard text. Using the `TextMode` property, this control can be set to display text in a single line, across multiple lines, or to hide the text being entered (for instance, in HTML password fields). The following code shows how we might use it in a simple login page:

```
<p>
  Username: <asp:TextBox id="userTextBox" TextMode="SingleLine"
      Columns="30" runat="server" />
</p>
<p>
  Password: <asp:TextBox id="passwordTextBox"
      TextMode="Password" Columns="30" runat="server" />
</p>
<p>
  Comments: <asp:TextBox id="commentsTextBox"
      TextMode="MultiLine" Columns="30" Rows="10"
      runat="server" />
</p>
```

In each of the instances above, the attribute `TextMode` dictates the kind of text box that's to be rendered.

HiddenField

`HiddenField` is a simple control that renders an `input` element whose `type` attribute is set to `hidden`. We can set its only important property, `Value`.

Button

By default, the `Button` control renders an `input` element whose `type` attribute is set to `submit`. When a button is clicked, the form containing the button is submitted to the server for processing, and both the `Click` and `Command` events are raised.

The following markup displays a `Button` control and a `Label`:

```
<asp:Button id="submitButton" Text="Submit" runat="server"
    OnClick="WriteText" />
<asp:Label id="messageLabel" runat="server" />
```

Notice the `OnClick` attribute on the control. When the button is clicked, the `Click` event is raised and the `WriteText` subroutine is called. The `WriteText` subroutine will contain the code that performs the intended function for this button, such as displaying a message to the user:

Visual Basic

```
Public Sub WriteText(s As Object, e As EventArgs)
  messageLabel.Text = "Hello World"
End Sub
```

C#
```
public void WriteText(Object s, EventArgs e)
{
  messageLabel.Text = "Hello World";
}
```

It's important to realize that events are associated with most web server controls, and the basic techniques involved in using them, are the same events and techniques we used with the `Click` event of the `Button` control. All controls implement a standard set of events because they all inherit from the `WebControl` base class.

ImageButton

An `ImageButton` control is similar to a `Button` control, but it uses an image that we supply in place of the typical system button graphic. Take a look at this example:

```
<asp:ImageButton id="myImgButton" ImageUrl="myButton.gif"
    runat="server" OnClick="WriteText" />
<asp:Label id="messageLabel" runat="server" />
```

The `Click` event of the `ImageButton` receives the coordinates of the point at which the image was clicked:

Visual Basic
```
Public Sub WriteText(s As Object, e As ImageClickEventArgs)
  messageLabel.Text = "Coordinate: " & e.X & "," & e.Y
End Sub
```

C#
```
public void WriteText(Object s, ImageClickEventArgs e)
{
  messageLabel.Text = "Coordinate: " + e.X + "," + e.Y;
}
```

LinkButton

A `LinkButton` control renders a hyperlink that fires the `Click` event when it's clicked. From the point of view of ASP.NET code, `LinkButtons` can be treated in much the same way as buttons, hence the name.

```
<asp:LinkButton id="myLinkButon" Text="Click Here"
    runat="server" />
```

HyperLink

The `HyperLink` control creates on your page a hyperlink that links to the URL in the `NavigateUrl` property. Unlike the `LinkButton` control, which offers features such as `Click` events and validation, `HyperLink`s are meant to be used to navigate from one page to the next.

```
<asp:HyperLink id="myLink" NavigateUrl="http://www.sitepoint.com/"
    ImageUrl="splogo.gif" runat="server">SitePoint</asp:HyperLink>
```

If it's specified, the `ImageUrl` attribute causes the control to display the specified image, in which case the text is demoted to acting as the image's alternate text.

CheckBox

You can use a `CheckBox` control to represent a choice that can have only two possible states—checked or unchecked.

```
<asp:CheckBox id="questionCheck" Text="I agree, I like .NET!"
    Checked="True" runat="server" />
```

The main event associated with a `CheckBox` is the `CheckChanged` event, which can be handled with the `OnCheckChanged` attribute. The `Checked` property is `True` if the checkbox is checked, and `False` otherwise.

RadioButton

A `RadioButton` is a lot like a `CheckBox`, except that `RadioButtons` can be grouped together to represent a set of options from which only one can be selected. Radio buttons are grouped together using the `GroupName` property.

```
<asp:RadioButton id="sanDiego" GroupName="City" Text="San Diego"
    runat="server" /><br />
<asp:RadioButton id="boston" GroupName="City" Text="Boston"
    runat="server" /><br />
<asp:RadioButton id="phoenix" GroupName="City" Text="Phoenix"
    runat="server" /><br />
<asp:RadioButton id="seattle" GroupName="City" Text="Seattle"
    runat="server" />
```

Like the `CheckBox` control, the main event associated with `RadioButtons` is the `CheckChanged` event, which can be handled with the `OnCheckChanged` attribute.

The other control we can use to display radio buttons is `RadioButtonList`, which we'll also meet in this chapter.

Image

An `Image` control creates an image that can be accessed dynamically from code; it equates to the `<img>` tag in HTML. Here's an example:

```
<asp:Image id="myImage" ImageUrl="mygif.gif" runat="server"
    AlternateText="description" />
```

ImageMap

The `ImageMap` control generates HTML to display images that have certain clickable regions called **hot spots**. Each hot spot reacts differently when clicked by the user.

These areas are defined using three controls that generate hot spots of different shapes: `CircleHotSpot`, `RectangleHotSpot`, and `PolygonHotSpot`. Here's an example that defines an image map with two circular hot spots:

```
<asp:ImageMap ID="myImageMap" runat="server" ImageUrl="image.jpg">
  <asp:CircleHotSpot AlternateText="Button1"
      Radius="20" X="50" Y="50" />
  <asp:CircleHotSpot AlternateText="Button2"
      Radius="20" X="100" Y="50" />
</asp:ImageMap>
```

Table 4.2. Possible values of `HotSpotMode`

`HotSpotMode` value	Behavior when hot spot is clicked
`Inactive`	none
`Navigate`	The user is navigated to the specified URL.
`NotSet`	When set for a `HotSpot`, the behavior is inherited from the parent `ImageMap`; if the parent `ImageMap` doesn't specify a default value, `Navigate` is set.
	When set for an `ImageMap`, this value is effectively equivalent to `Navigate`.
`PostBack`	The hot spot raises the `Click` event that can be handled server-side to respond to the user action.

To configure the action that results when a hot spot is clicked by the user, we set the `HotSpotMode` property of the `ImageMap` control, or the `HotSpotMode` property of the individual hot spot objects, or both, using the values shown in Table 4.2. If the `HotSpotMode` property is set for the `ImageMap` control as well as for an individual hot spot, the latter property will override that set for the more general `ImageMap` control.

The Microsoft .NET Framework 2.0 SDK Documentation for the `ImageMap` class and HotSpotMode enumeration contains detailed examples of the usage of these values.

PlaceHolder

The `PlaceHolder` control lets us add elements at a particular place on a page at any time, dynamically, through our code.

```
<asp:PlaceHolder id="placeHolder" runat="server" />
```

The following code dynamically adds a new `HtmlButton` control within the placeholder:

Visual Basic

```
Public Sub Page_Load()
  Dim button myButton As HtmlButton = New HtmlButton()
  myButton.InnerText = "My New Button"
  placeHolder.Controls.Add(myButton)
End Sub
```

C#

```
public void Page_Load()
{
  HtmlButton button myButton = new HtmlButton();
  myButton.InnerText = "My New Button";
  placeHolder.Controls.Add(myButton);
}
```

Panel

The `Panel` control functions similarly to the `div` element in HTML, in that it allows the set of items that resides within the tag to be manipulated as a group. For instance, the `Panel` could be made visible or hidden by a `Button`'s `Click` event:

```
<asp:Panel id="myPanel" runat="server">
  <p>Username: <asp:TextBox id="usernameTextBox" Columns="30"
      runat="server" /></p>
  <p>Password: <asp:TextBox id="passwordTextBox"
      TextMode="Password" Columns="30" runat="server" />
  </p>
</asp:Panel>
<asp:Button id="hideButton" Text="Hide Panel" OnClick="HidePanel"
    runat="server" />
```

The code above places two `TextBox` controls within a `Panel` control. The `Button` control is outside of the panel. The `HidePanel` subroutine would then control the `Panel`'s visibility by setting its `Visible` property to `False`:

Visual Basic

```
Public Sub HidePanel(s As Object, e As EventArgs)
  myPanel.Visible = False
End Sub
```

C#

```
public void HidePanel(Object s, EventArgs e)
{
  myPanel.Visible = false;
}
```

In this case, when the user clicks the button, the `Click` event is raised and the `HidePanel` subroutine is called, which sets the `Visible` property of the `Panel` control to `False`.

List Controls

Here, we'll meet the ASP.NET controls that display simple lists of elements: `ListBox`, `DropDownList`, `CheckBoxList`, `RadioButtonList`, and `BulletedList`.

DropDownList

A `DropDownList` control is similar to the HTML `select` element. The `DropDownList` control allows you to select one item from a list using a drop-down menu.

```
<asp:DropDownList id="ddlFavColor" runat="server">
  <asp:ListItem Text="Red" value="red" />
  <asp:ListItem Text="Blue" value="blue" />
```

```
  <asp:ListItem Text="Green" value="green" />
</asp:DropDownList>
```

The most useful event that this control provides is `SelectedIndexChanged`. This event is exposed by other list controls, such as the `CheckBoxList` and `RadioButtonList` controls, allowing for easy programmatic interaction with the control. These controls can also be bound to a database, allowing you to extract dynamic content into a drop-down menu.

ListBox

A `ListBox` control equates to the HTML `select` element with either the `multiple` or `size` attribute set (`size` would need to be set to a value of 2 or more). If you set the `SelectionMode` attribute to `Multiple`, the user will be able to select more than one item from the list, as in this example:

```
<asp:ListBox id="listTechnologies" runat="server"
    SelectionMode="Multiple">
  <asp:ListItem Text="ASP.NET" Value="aspnet" />
  <asp:ListItem Text="JSP" Value="jsp" />
  <asp:ListItem Text="PHP" Value="php" />
  <asp:ListItem Text="CGI" Value="cgi" />
  <asp:ListItem Text="ColdFusion" Value="cf" />
</asp:ListBox>
```

RadioButtonList

Like the `RadioButton` control, the `RadioButtonList` control represents radio buttons. However, the `RadioButtonList` control represents a list of radio buttons and uses more compact syntax. Here's an example:

```
<asp:RadioButtonList id="favoriteColor" runat="server">
  <asp:ListItem Text="Red" Value="red" />
  <asp:ListItem Text="Blue" Value="blue" />
  <asp:ListItem Text="Green" Value="green" />
</asp:RadioButtonList>
```

CheckBoxList

As you may have guessed, the `CheckBoxList` control represents a group of check boxes; it's equivalent to using several `CheckBox` controls in row:

```
<asp:CheckBoxList id="favoriteFood" runat="server">
  <asp:ListItem Text="Pizza" Value="pizza" />
```

```
  <asp:ListItem Text="Tacos" Value="tacos" />
  <asp:ListItem Text="Pasta" Value="pasta" />
</asp:CheckBoxList>
```

BulletedList

The `BulletedList` control displays bulleted or numbered lists, using `<ul>` (unordered list) or `<ol>` (ordered list) tags. Unlike the other list controls, the `BulletedList` doesn't allow the selection of items, so the `SelectedIndexChanged` event isn't supported.

The first property you'll want to set is `DisplayMode`, which can be `Text` (the default), or `HyperLink`, which will render the list items as links. When `DisplayMode` is set to `HyperLink`, you can use the `Click` event to react when the user clicks on one of the items.

The other important property is `BulletStyle`, which determines the style of the bullets. The accepted values are `Numbered` (1, 2, 3, …), `LowerAlpha` (a, b, c, …), `UpperAlpha` (A, B, C, …), `LowerRoman` (i, ii, iii, …), `UpperRoman` (I, II, III, …), `Circle`, `Disc`, `Square`, and `CustomImage`. If the style is set to `CustomImage`, you'll also need to set the `BulletStyleImageUrl` to specify the image to be used for the bullets. If the style is one of the numbered lists, you can also set the `FirstBulletNumber` property to specify the first number or letter that's to be generated.

Advanced Controls

These controls are advanced in terms of their usage, the HTML code they generate, and the background work they do for you. Some of these controls aren't available to older versions of ASP.NET; we'll learn more about many of them (as well as others that aren't covered in this chapter) as we progress through this book.

Calendar

The `Calendar` is a great example of the reusable nature of ASP.NET controls. The `Calendar` control generate the markup to display an intuitive calendar in which the user can click to select or move between days, weeks, months, and so on.

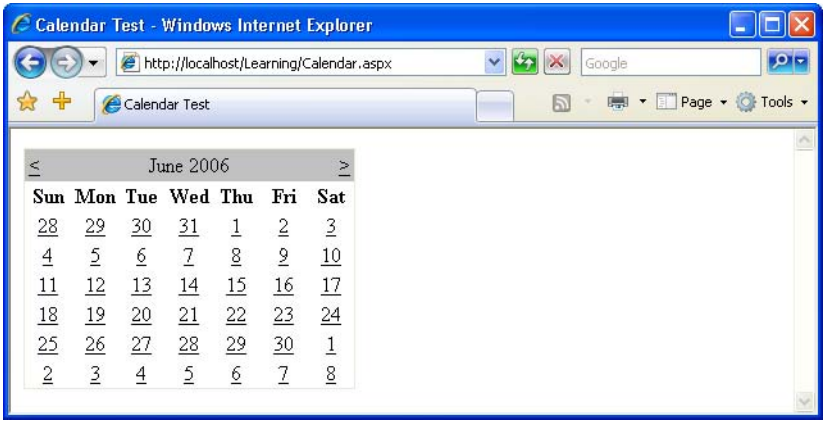
The `Calendar` control requires very little customization, and can be created within a page like this:

File: **Calendar.aspx (excerpt)**

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
    "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <title>Calendar Test</title>
  </head>
  <body>
    <form runat="server">
      <asp:Calendar id="myCalendar" runat="server" />
    </form>
  </body>
</html>
```

If you save this page in the `Learning` folder and load it, you'd get the output shown in Figure 4.4.

Figure 4.4. Displaying the default calendar



The `Calendar` control contains a wide range of properties, methods, and events, including those listed in Table 4.3.

Table 4.3. Some of the `Calendar` control's properties

Property	Description
`DayNameFormat`	This property sets the format of the day names. Its possible values are `FirstLetter`, `FirstTwoLetters`, `Full`, and `Short`. The default is `Short`, which displays the three-letter abbreviation.
`FirstDayOfWeek`	This property sets the day of the week that begins each week in the calendar. By default, the value of this property is determined by your server's region settings, but you can set this to `Sunday` or `Monday` if you want to control it.
`NextPrevFormat`	Set to `CustomText` by default, this property can be set to `ShortMonth` or `FullMonth` to control the format of the next and previous month links.
`SelectedDate`	This property contains a DateTime value that specifies the highlighted day. You'll use this property a lot to determine which day the user has selected.
`SelectionMode`	This property determines whether days, weeks, or months can be selected; its possible values are `Day`, `DayWeek`, `DayWeekMonth`, and `None`, and the default is `Day`. When `Day` is selected, a user can only select a day; when `DayWeek` is selected, a user can select a day or an entire week; and so on.
`SelectMonthText`	This property controls the text of the link that's displayed to allow users to select an entire month from the calendar.
`SelectWeekText`	This property controls the text of the link that's displayed to allow users to select an entire week from the calendar.
`ShowDayHeader`	If `True`, this property displays the names of the days of the week. The default is `True`.
`ShowGridLines`	If `True`, this property renders the calendar with grid lines. The default is `True`.
`ShowNextPrevMonth`	If `True`, this property displays next/previous month links. The default is `True`.
`ShowTitle`	If `True`, this property displays the calendar's title. The default is `False`.

Property	Description
`TitleFormat`	This property determines how the month name appears in the title bar. Possible values are `Month` and `MonthYear`. The default is `MonthYear`.
`TodaysDate`	This DateTime value sets the calendar's current date. By default, this value is not highlighted within the `Calendar` control.
`VisibleDate`	This DateTime value controls which month is displayed.

Let's take a look at an example that uses some of these properties, events, and methods to create a `Calendar` control that allows users to select days, weeks, and months. Modify the calendar in `Calendar.aspx`, and add a label to it, as follows:

File: **`Calendar.aspx` (excerpt)**

```
<asp:Calendar ID="myCalendar" runat="server" DayNameFormat="Short"
    FirstDayOfWeek="Sunday" NextPrevFormat="FullMonth"
    SelectionMode="DayWeekMonth" SelectWeekText="Select Week"
    SelectMonthText="Select Month" TitleFormat="Month"
    OnSelectionChanged="SelectionChanged" />
<h2>You selected these dates:</h2>
<asp:Label ID="myLabel" runat="server" />
```

Now add a `<script runat="server">` tag to the head of the web form to include the `SelectionChanged` event handler referenced by your calendar:

Visual Basic File: **`Calendar.aspx` (excerpt)**

```
<script runat="server" language="VB">
  Sub SelectionChanged(ByVal s As Object, ByVal e As EventArgs)
    myLabel.Text = ""
    For Each d As DateTime In myCalendar.SelectedDates
      myLabel.Text &= d.ToString("D") & "<br />"
    Next
  End Sub
</script>
```

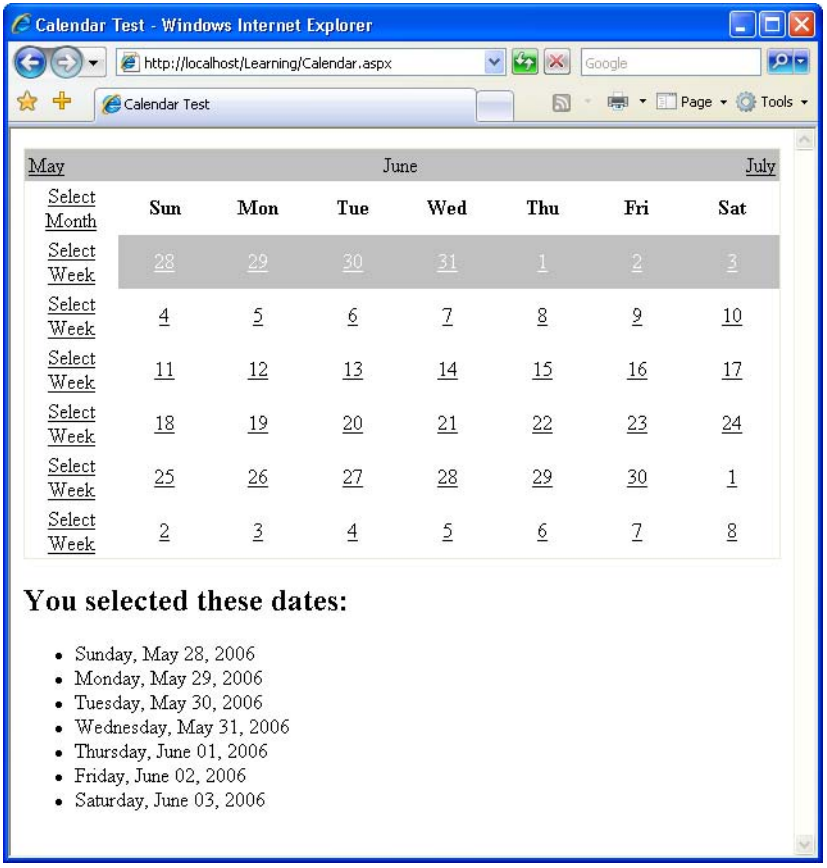
C# File: **`Calendar.aspx` (excerpt)**

```
<script runat="server" language="C#">
  void SelectionChanged(Object s, EventArgs e)
  {
    myLabel.Text = "";
    foreach (DateTime d in myCalendar.SelectedDates)
    {
      myLabel.Text += d.ToString("D") + "<br />";
```

```
    }
  }
</script>
```

Save your work and test it in a browser. Try selecting a day, week, or month. The selection will be highlighted similar to this display shown in Figure 4.5.

Figure 4.5. Using the `Calendar` control



In `SelectionChanged`, we loop through each date that the user has selected, and append it to the `BulletedList` we added to the page.

AdRotator

The `AdRotator` control allows you to display a list of banner advertisements within your web application at random. However, it's more than a mere substitute for creating a randomization script from scratch. Since the `AdRotator` control gets its content from an XML file, the administration and updating of banner advertisement files and their properties is a snap. Also, the XML file allows you to control the banner's image, link, link target, and frequency of appearance in relation to other banner ads.

The benefits of using this control don't stop there, though. As most of the `AdRotator` control's properties reside within an XML file, if you wished, you could share that XML file on the Web, essentially allowing value added resellers (VARS), or possibly your companies' partners, to use your banner advertisements on their web sites.

XML Basics

In essence, XML is simply a text-based format for the transfer or storage of data; it contains no details on how that data should be presented. XML is very easy to start with because of its close resemblance to your old friend HTML: both are largely comprised of tags inside angle brackets (< and >), and any tag may contain attributes specific to that tag. The biggest difference between XML and HTML is that, rather than providing a fixed set of tags as HTML does, XML allows us to create our own tags to describe the data we wish to represent.

Take a look at the following HTML element:

```
<p>Star Wars Episode I: The Phantom Menace</p>
```

This example describes the content between the tags as a paragraph. This is fine if all we are concerned with is displaying the words "Star Wars Episode I: The Phantom Menace" on a web page. But what if we want to access those words as data?

Like HTML, XML's purpose is to describe the content of a document. But unlike HTML, XML doesn't describe how that content should be displayed; it describes *what the content is*. Using XML, the web author can mark up the contents of a document, describing that content in terms of its relevance as data.

We can use XML to mark up the words "Star Wars Episode I: The Phantom Menace" in a way that better reflects this content's significance as data:

```
<film>
  <title>Star Wars Episode I: The Phantom Menace</title>
</film>
```

Here, the XML tag names we've chosen best describe the contents of the element. We also define our own attribute names as necessary. For instance, in the example above, you may decide that you want to differentiate between the VHS version and the DVD version of the film, or record the name of the movie's director. This can be achieved by adding attributes and elements, as shown below:

```
<film format="DVD">
  <title>Star Wars Episode I: The Phantom Menace</title>
  <director>George Lucas</director>
</film>
```

If you want to test this out, create a file called `ads.xml` in your `Learning` folder, and insert the content presented below. Feel free to create your own banners, or to use those provided in the code archive for this book.

File: **`Ads.xml` (excerpt)**

```
<Advertisements>
  <Ad>
    <ImageUrl>workatdorknozzle.gif</ImageUrl>
    <NavigateUrl>http://www.dorknozzle.com</NavigateUrl>
    <TargetUrl>_blank</TargetUrl>
    <AlternateText>Work at Dorknozzle.com!</AlternateText>
    <Keyword>HR Sites</Keyword>
    <Impressions>2</Impressions>
  </Ad>
  <Ad>
    <ImageUrl>getthenewsletter.gif</ImageUrl>
    <NavigateUrl>http://www.dorknozzle.com</NavigateUrl>
    <TargetUrl>_blank</TargetUrl>
    <AlternateText>Get the Nozzle Newsletter!</AlternateText>
    <Keyword>Marketing Sites</Keyword>
    <Impressions>1</Impressions>
  </Ad>
</Advertisements>
```

As you can see, the `Advertisements` element is the root node, and in accordance with the XML specification, it appears only once. For each individual advertisement, we simply add an `Ad` child element. For instance, the above advertisement file contains details for two banner advertisements.

As you've probably noticed by now, the `.xml` file enables you to specify properties for each banner advertisement by inserting appropriate elements inside each of the `Ad` elements. These elements include:

`ImageURL`
: the URL of the image to display for the banner ad

`NavigateURL`
: the web page to which your users will navigate when they click the banner ad

`AlternateText`
: the alternative text to display for browsers that do not support images

`Keyword`
: the keyword to use to categorize your banner ad

  If you use the `KeywordFilter` property of the `AdRotator` control, you can specify the categories of banner ads to display.

`Impressions`
: the relative frequency that a particular banner ad should be shown in relation to other banner advertisements

  The higher this number, the more frequently that specific banner will display in the browser. The number provided for this element can be as low as one, but cannot exceed 2,048,000,000; if it does, the page throws an exception.

Except for `ImageURL`, all these elements are optional. Also, if you specify an `Ad` without a `NavigateURL`, the banner ad will display without a hyperlink.

To make use of this `Ads.xml` file, create a new ASP.NET page, called `AdRotator.aspx`, with the following code:

File: **`AdRotator.aspx` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>AdRotator Control</title>
  </head>
  <body>
    <form runat="server">
      <asp:AdRotator ID="adRotator" runat="server"
```

```
            AdvertisementFile="Ads.xml" />
    </form>
  </body>
</html>
```

You'll need to download `workatdorknozzle.gif` and `getthenewsletter.gif` and place them in the `Learning` folder in order to see these ad images. Save your work and test it in the browser; the display should look something like Figure 4.6.

Figure 4.6. Displaying ads using `AdRotator.aspx`

Refresh the page a few times, and you'll notice that the first banner appears more often than the second. This occurs because the `Impression` value for the first `Ad` is double the value set for the second banner, so it will appear twice as often.

`TreeView`

The `TreeView` control is a very powerful control that's capable of displaying a complex hierarchical structure of items. Typically we'd use it to view a directory structure or a site navigation hierarchy, but it could be used to display a family tree, a corporate organizational structure, or any other hierarchical structure.

The `TreeView` can pull its data from various sources. You'll learn more about the various kinds of data sources later in the book; here, we'll focus on the `SiteMapDataSource` class, which, as its name suggests, contains a hierarchical sitemap. By default, this sitemap is read from a file called `Web.sitemap` located in the root of your project. The `Web.sitemap` file is an XML file that looks like this:

File: **Web.sitemap**

```
<siteMap
    xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode title="Home" url="~/Default.aspx"
      description="Home">
    <siteMapNode title="TreeViewDemo" url="~/TreeViewDemo.aspx"
        description="TreeView Example" />
    <siteMapNode title="ClickEvent" url="~/ClickEvent.aspx"
        description="ClickEvent Example" />
    <siteMapNode title="Loops"  url="~/Loops.aspx"
        description="Loops Example" />
  </siteMapNode>
</siteMap>
```

Web.sitemap Limitation

An important limitation to note when working with `Web.sitemap` files is that they must contain only one `siteMapNode` as the direct child of the root `siteMap` element.

In the example above, the `siteMapNode` with the title `Home` is this single `siteMapNode`. If we added another `siteMapNode` alongside (rather than inside) this element, the `Web.sitemap` file would no longer be valid.

To use this file, you'll need to add a `SiteMapDataSource` control to the page, as well as a `TreeView` control that uses this data source, like this:

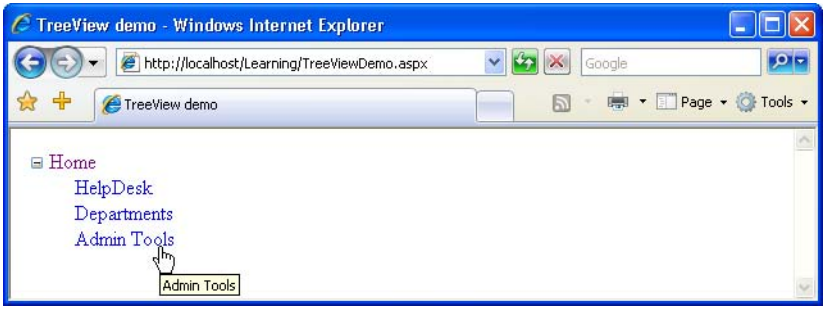
File: **TreeViewDemo.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>TreeView Demo</title>
  </head>
  <body>
    <form runat="server">
      <asp:SiteMapDataSource ID="mySiteMapDataSource"
          runat="server" />
      <asp:TreeView ID="myTreeView" runat="server"
          DataSourceID="mySiteMapDataSource" />
    </form>
  </body>
</html>
```

Note that although the `SiteMapDataSource` is a control, it does not generate any HTML within the web page. There are many data source controls like this; we'll delve into this in more detail later.

When combined with the example `Web.sitemap` file above, this web form would generate an output such as that shown in Figure 4.7.

Figure 4.7. A simple `TreeView` control



As you can see, the `TreeView` control generated the tree for us. The root Home node can even be collapsed or expanded.

In many cases, we don't want to show the root node; we can hide it from view by setting the `ShowStartingNode` property of the `SiteMapDataSource` to `false`:

```
<asp:SiteMapDataSource ID="mySiteMapDataSource" runat="server"
    ShowStartingNode="false" />
```

SiteMapPath

The `SiteMapPath` control provides the functionality to generate a **breadcrumb** navigational structure for your site. Breadcrumb systems help to orientate users, giving them an easy way to identify their current location within the site, and provide handy links to the current location's ancestor nodes. An example of a breadcrumb navigation system is shown in Figure 4.8.

The `SiteMapPath` control will automatically use any `SiteMapDataSource` control that exists in a web form to display a user's current location within the site. For example, you could simply add the following code to the form we worked with in the previous example to achieve the effect shown in Figure 4.8:

Figure 4.8. A breadcrumb created using the `SiteMapPath` control

File: **`TreeViewDemo.aspx` (excerpt)**

```
<asp:SiteMapPath id="mySiteMapPath" runat="server"
    PathSeparator=" > ">
</asp:SiteMapPath>
```

If you run the example now, you'll see the breadcrumb appear exactly as it's shown in Figure 4.8.

Note that the `SiteMapPath` control shows only the nodes that correspond to existing pages of your site, so if you don't have a file named `Default.aspx`, the root node link won't show up. Similarly, if the page you're loading isn't named `TreeViewDemo.aspx`, the `SiteMapPath` control won't generate any output.

`Menu`

The `Menu` control is similar to `TreeView` in that it displays hierarchical data from a data source; the ways in which we work with both controls are also very similar. The most important differences between the two lie in their appearances, and the fact that `Menu` supports templates for better customization and displays only two levels of items (menu and submenu items).

`MultiView`

The `MultiView` control is similar to `Panel` in that it doesn't generate interface elements itself, but contains other controls. A `MultiView` can store more pages of data (called **views**), and lets you show one page at a time. You can change the active view (the one being presented to the visitor) by setting the value of the

`ActiveViewIndex` property. The first page corresponds to an `ActiveViewIndex` of `0`, the second page is `1`, the third page is `2`, and so on.

The contents of each template are defined inside child `View` elements. Consider the following code snippet, which creates a `Button` control, and a `MultiView` control:

File: **`MultiViewDemo.aspx` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>MultiView Demo</title>
  </head>
  <body>
    <form runat="server">
      <p>
        <asp:Button id="myButton" Text="Switch Page"
            runat="server" OnClick="SwitchPage" />
      </p>
      <asp:MultiView ID="myMultiView" runat="server"
          ActiveViewIndex="0">
        <asp:View ID="firstView" runat="server">
          <p>... contents of the first view ...</p>
        </asp:View>
        <asp:View ID="secondView" runat="server">
          <p>... contents of the second view ...</p>
        </asp:View>
      </asp:MultiView>
    </form>
  </body>
</html>
```

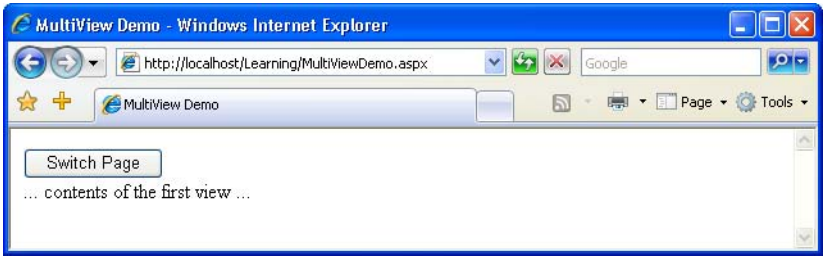
As you can see, by default, the `ActiveViewIndex` is `0`, so when this code is first executed, the `MultiView` will display its first template, shown in Figure 4.9.

Clicking on the button will cause the second template to be displayed. Here's the code for the `SwitchPage` event handler:

Visual Basic File: **`MultiViewDemo.aspx` (excerpt)**

```
<script runat="server" language="VB">
  Sub SwitchPage(s as Object, e as EventArgs)
    myMultiView.ActiveViewIndex = _
        (myMultiView.ActiveViewIndex + 1) Mod 2
```

Figure 4.9. Using the `MultiView` control




```
  End Sub
</script>
```

C# File: **`MultiViewDemo.aspx` (excerpt)**

```
<script runat="server" language="C#">
  public void SwitchPage(Object s, EventArgs e)
  {
    myMultiView.ActiveViewIndex =
        (myMultiView.ActiveViewIndex + 1) % 2;
  }
</script>
```

This simple subroutine uses the modulo operator to set the `ActiveViewIndex` to `1` when its original value is `0`, and vice versa.

The `MultiView` controls has a number of other handy features, so be sure to check the documentation for this control if you're using it in a production environment.

`Wizard`

The `Wizard` control is a more advanced version of the `MultiView` control. It's able to display one or more pages at a time, but also includes additional built-in functionality such as navigation buttons, and a sidebar that displays the wizard's steps.

`FileUpload`

The `FileUpload` control allows you to let visitors upload files to your server. You'll learn how to use this control in Chapter 14.

Web User Controls

As you build real-world projects, you'll frequently encounter pieces of the user interface that appear in multiple places—headers or footers, navigation links, and login boxes are just a few examples. Packaging their forms and behaviors into your own controls will allow you to reuse these components just as you can reuse ASP.NET's built-in controls.

Building your own web server controls involves writing advanced VB or C# code, and is not within the scope of this book, but it's good to know that it's possible. Creating customized web server controls makes sense when you need to build more complex controls that provide a high level of control and performance, or you want to create controls that can be integrated easily into many projects.

Those of us without advanced coding skills can develop our own controls by creating **web user controls**. These are also powerful and reusable within a given project; they can expose properties, events, and methods, just like other controls; and they're easy to implement.

A web user control is represented by a class that inherits from `System.Web.UI.UserControl`, and contains the basic functionality that you need to extend to create your own controls. The main drawback to using web user controls is that they're tightly integrated into the projects in which they're implemented. As such, it's more difficult to distribute them, or include them in other projects, than it is to distribute or reuse web server controls.

Web user controls are implemented very much like normal web forms—they're comprised of other controls, HTML markup, and server-side code. The file extension of a web user control is `.ascx`.

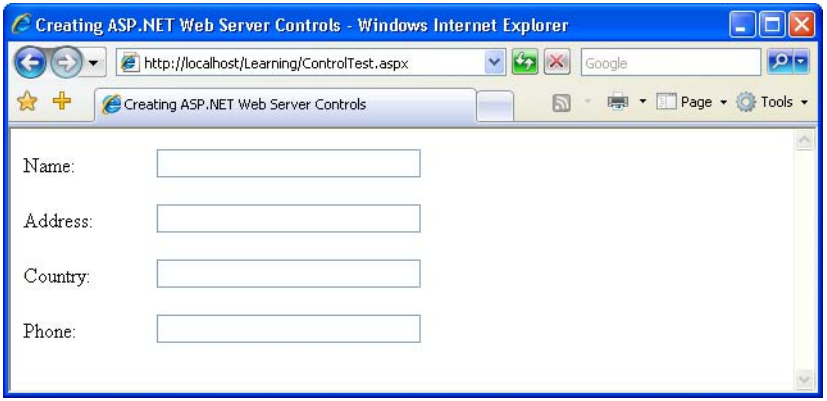
Creating a Web User Control

Let's get a feel for web user controls by stepping through a simple example. Let's say that in your web site, you have many forms consisting of pairs of `Label` and `TextBox` controls, like the one shown in Figure 4.10.

All the labels must have a fixed width of 100 pixels, and the text boxes must accept a maximum of 20 characters.

Rather than adding many labels and text boxes to the form, and then having to set all their properties, let's make life easier by building a web user control that

Figure 4.10. A simple form



includes a `Label` of the specified width, and a `TextBox` that accepts 20 characters; you'll then be able to reuse the web user control wherever it's needed in your project.

In your `Learning` folder, create a new file named `SmartBox.ascx`. Then, add the control's constituent controls—a `Label` control and a `TextBox` control—as shown below:

File: **`SmartBox.ascx` (excerpt)**

```
<p>
  <asp:Label ID="myLabel" runat="server" Text="" Width="100" />
  <asp:TextBox ID="myTextBox" runat="server" Text="" Width="200"
      MaxLength="20" />
</p>
```

Label Widths in Firefox

Unfortunately, setting the `Width` property of the `Label` control doesn't guarantee that the label will appear at that width in all browsers. The current version of Firefox, for example, will not display the above label in the way it appears in Internet Explorer.

To get around this, you should use a CSS style sheet and the `CssClass` property, which we'll take a look at later in this chapter.

In Chapter 3 we discussed properties briefly, but we didn't explain how you could create your own properties within your own classes. So far, you've worked with

many properties of the built-in controls. For example, you've seen a lot of code that sets the `Text` property of the `Label` control.

As a web user control is a class, it can also have methods, properties, and so on. Our `SmartBox` control extends the base `System.Web.UI.UserControl` class by adding two properties:

- `LabelText` is a write-only property that allows the forms using the control to set the control's label text.
- `Text` is a read-only property that returns the text typed by the user in the text box.

Let's add a server-side `script` element that will give our control two properties — one called `Text`, for the text in the `TextBox`, and one called `LabelText`, for the text in the `Label`:

Visual Basic File: **SmartBox.ascx (excerpt)**

```
<script runat="server" language="VB">
  Public WriteOnly Property LabelText() As String
    Set(ByVal value As String)
      myLabel.Text = value
    End Set
  End Property

  Public ReadOnly Property Text() As String
    Get
      Text = myTextBox.Text
    End Get
  End Property
</script>
```

C# File: **SmartBox.ascx (excerpt)**

```
<script runat="server" language="C#">
  public string LabelText
  {
    set
    {
      myLabel.Text = value;
    }
  }
  public string Text
  {
    get
    {
```

```
      return myTextBox.Text;
    }
  }
</script>
```

Just like web forms, web user controls can work with code-behind files, but, in an effort to keep our examples simple, we aren't using them here. You'll meet more complex web user controls in the chapters that follow.

When you use the `SmartBox` control in a form, you can set its label and have the text entered by the user, like this:

Visual Basic

```
mySmartBox.LabelText = "Address:"
userAddress = mySmartBox.Text
```

C#

```
mySmartBox.LabelText = "Address:";
userAddress = mySmartBox.Text;
```

Let's see how we implemented this functionality. In .NET, properties can be read-only, write-only, or read-write. In many cases, you'll want to have properties that can be both read and write, but in this case, we want to be able to set the text of the inner `Label`, and to read the text from the `TextBox`.

To define a write-only property in VB, you need to use the `WriteOnly` modifier. Write-only properties need only define a special block of code that starts with the keyword `Set`. This block of code, called an **accessor**, is just like a subroutine that takes as a parameter the value that needs to be set. The block of code uses this value to perform the desired action—in the case of the `LabelText` property, that action sets the `Text` property of our `Label` control, as shown below:

Visual Basic File: **`SmartBox.ascx` (excerpt)**

```
Public WriteOnly Property LabelText() As String
  Set(ByVal value As String)
    myLabel.Text = value
  End Set
End Property
```

Assuming that a form uses a `SmartBox` object called `mySmartBox`, we could set the `Text` property of the `Label` like this:

Visual Basic

```
mySmartBox.LabelText = "Address:"
```

When this code is executed, the `Set` accessor of the `LabelText` property is executed with its *`value`* parameter set to `Address:`. The `Set` accessor uses this value to set the `Text` property of the `Label`.

The other accessor you can use when defining properties is `Get`; this allows us to read values instead of writing them. Obviously, you aren't allowed to add a `Get` accessor to a `WriteOnly` property, but one is required for a `ReadOnly` property, such as `Text`:

Visual Basic — File: **SmartBox.ascx (excerpt)**

```
Public ReadOnly Property Text() As String
  Get
    Text = myTextBox.Text
  End Get
End Property
```

The `Text` property is `ReadOnly`, but it doesn't need to be. If you wanted to allow the forms using the control to set some default text to the `TextBox`, you'd need to add a `Set` accessor, and remove the `ReadOnly` modifier.

When defining a property in C#, you don't need to set any special modifiers, such as `ReadOnly` or `WriteOnly`, for read-only or write-only properties. A property that has only a `get` accessor will, by default, be considered read-only:

C# — File: **SmartBox.ascx (excerpt)**

```
public string Text
{
  get
  {
    return myTextBox.Text;
  }
}
```

Likewise, a property that has only a `set` accessor will be considered to be write-only:

C# — File: **SmartBox.ascx (excerpt)**

```
public string LabelText
{
  set
  {
    myLabel.Text = value;
  }
}
```

Using the Web User Control

Once the user control has been created, it can be referenced from any ASP.NET page using the `Register` directive, as follows:

```
<%@ Register TagPrefix="prefix" TagName="name"
    Src="source.ascx" %>
```

The `Register` directive requires three attributes:

`TagPrefix`
: the prefix for the user control, which allows you to group related controls together, and avoid naming conflicts

`TagName`
: the control's tag name, which will be used when the control is added to the ASP.NET page

`Src`
: the path to the `.ascx` file that describes the user control

After registering the control, we create instances of it using the *<TagPrefix:TagName>* format. Let's try an example that uses the `SmartBox` control. Create a new file named `ControlTest.aspx` in your `Learning` folder, and give it this content:

File: **`ControlTest.aspx` (excerpt)**

```
<%@ Register TagPrefix="sp" TagName="SmartBox"
    Src="SmartBox.ascx" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Creating ASP.NET Web Server Controls</title>
  </head>
  <body>
    <form id="Form1" runat="server">
      <sp:SmartBox id="nameSb" runat="server" LabelText="Name:" />
      <sp:SmartBox id="addressSb" runat="server"
          LabelText="Address:" />
      <sp:SmartBox id="countrySb" runat="server"
          LabelText="Country:" />
      <sp:SmartBox id="phoneSb" runat="server"
          LabelText="Phone:" />
    </form>
```

```
  </body>
</html>
```

Loading this page will produce the output we saw in Figure 4.10.

Now, this is a very simple example indeed, but we can easily extend it for other purposes. You can see in the code snippet that we set the `LabelText` property directly in the control's tag; we could have accessed the properties from our code instead. Here's an example:

Visual Basic — File: **ControlTest.aspx (excerpt)**

```
<script runat="server" language="VB">
  Protected Sub Page_Load()
    nameSb.LabelText = "Name:"
    addressSb.LabelText = "Address:"
    countrySb.LabelText = "Country:"
    phoneSb.LabelText = "Phone:"
  End Sub
</script>
```

C# — File: **ControlTest.aspx (excerpt)**

```
<script runat="server" language="C#">
  protected void Page_Load()
  {
    nameSb.LabelText = "Name:";
    addressSb.LabelText = "Address:";
    countrySb.LabelText = "Country:";
    phoneSb.LabelText = "Phone:";
  }
</script>
```

Master Pages

Master pages are a new feature of ASP.NET 2.0 that can make an important difference in the way we compose web forms. Master pages are similar to web user controls in that they are also composed of HTML and other controls; they can be extended with the addition of events, methods, or properties; and they can't be loaded directly by users—instead, they're used as building blocks to design the structure of your web forms.

A master page is a page template that can be applied to give many web forms a consistent appearance. For example, a master page can set out a standard structure

containing the header, footer, and other elements that you expect to display in multiple web forms within a web application.

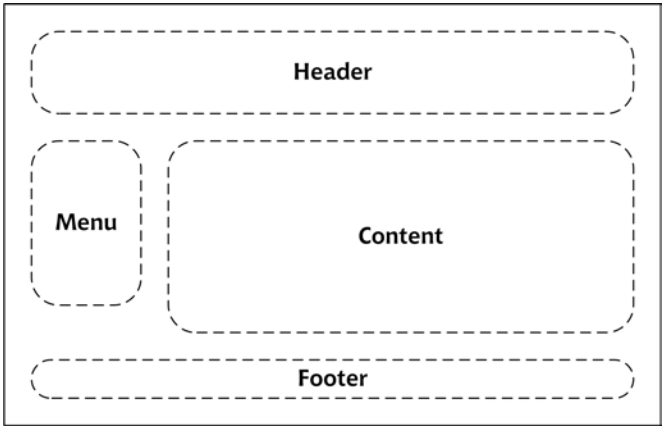
Master page files have the `.master` extension, and, just like web forms and web user controls, they support code-behind files. All master pages inherit from the class `System.Web.UI.MasterPage`.

Designing a site structure using master pages and web user controls gives you the power to easily modify and extend the site. If your site uses these features in a well-planned way, it can be very easy to modify certain details in the layout or functionality of your site, because updating a master page or a web user control has immediate effects on all the web forms that use the file.

As we've already mentioned, a master page is built using HTML and controls, including the special `ContentPlaceHolder` control. As its name suggests, the `ContentPlaceHolder` is a placeholder that can be filled with content relevant to the needs of each web form that uses the master page. In creating a master page, we include all of the basic structure of future pages in the master page itself, including the `<html>`, `<head>`, and `<body>` tags, and let the web forms specify the content that appears in the placeholders.

Let's see how this works with a simple example. Suppose we have a site which has many pages that contain a standard header, footer, and navigation menu, laid out as per the wireframe shown in Figure 4.11.

Figure 4.11. A simple web site layout



If all the pages in the site have the same header, footer, and navigation menu, it makes sense to include these components in a master page, and to build several web forms that customize only the content areas on each page. We'll begin to create such a site in Chapter 5, but let's work through a quick example here.

To keep this example simple, we won't include a menu here: we'll include just the header, the footer, and the content placeholder. In your `Learning` folder, create a new file named `FrontPages.master`, and write the following code into it:

File: **`FrontPages.master` (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Front Page</title>
  </head>
  <body>
    <form id="myForm" runat="server">
      <h1>Welcome to SuperSite Inc!</h1>
      <asp:ContentPlaceHolder id="FrontPageContent"
          runat="server" />
      <p>Copyright 2006</p>
    </form>
  </body>
</html>
```

The master page looks almost like a web form, except for one important detail: it has an empty `ContentPlaceHolder` control. If you want to build a web form based on this master page, you just need to reference the master page using the `Page` directive in the web form, and add a `Content` control that includes the content you want to insert.

Let's try it. Create a web form called `FrontPage.aspx`, and add this code to it:

File: **`FrontPage.aspx` (excerpt)**

```
<%@ Page MasterPageFile="FrontPages.master" %>
<asp:Content id="myContent" runat="server"
    ContentPlaceHolderID="FrontPageContent">
  <p>
    Welcome to our web site! We hope you'll enjoy your visit.
  </p>
</asp:Content>
```

You're all set now! Loading `FrontPage.aspx` in your browser will generate the output shown in Figure 4.12.

Figure 4.12. Using a master page

Although the example is simplistic, it's easy to see the possibilities: you can create many web forms based on this template very easily. In our case, the master page contains a single `ContentPlaceHolder`, but it could have more. Also, the master page can define some default content for display inside the `ContentPlaceHolder` on pages whose web forms don't provide a `Content` element for that placeholder.

Using Cascading Style Sheets (CSS)

It's clear that controls make it easy for us to reuse pieces of functionality in multiple places. For example, I can't imagine an easier way to add calendars to many web forms than to use the `Calendar` web server control.

However, controls don't solve the problem of defining and managing the visual elements of your web site. Modern web sites need constant updating to keep them fresh, and it's not much fun editing hundreds of pages by hand just to change a border color, for example, and then having to check everything to ensure that the changes are consistent. The process is even more painful if the client wants a more serious update, like rearranging components on the pages.

The good news is that this maintenance work can be made a lot easier by planning ahead, correctly following a few basic rules, and efficiently using the tools HTML and ASP.NET offer you.

An essential tool for building reusable visual styles is **CSS** (Cascading Style Sheets). HTML was initially designed to deliver simple text content, and paid

little attention to the specifics of how particular items appeared in the browser. HTML left it to the individual browsers to work out these intricacies, and tailor the output to the limitations and strengths of users' machines. While we can change font styles, sizes, colors, and so on using HTML tags, this practice can lead to verbose code and pages that are very hard to restyle at a later date.

CSS gives web developers the power to create one set of styles in a single location, and to apply those styles to all of the pages in our web site. All the pages to which the style sheet is applied will display the same fonts, colors, and sizes, giving the site a consistent feel. Regardless of whether our site contains three pages or 300, when we alter the styles in the style sheet, our changes are immediately applied to all pages that use the style sheet.

Look out for Themes and Skins

ASP.NET 2.0 provides extra value and power to those building reusable visual elements through its offerings of **themes** and **skins**. You'll learn more about these features in Chapter 5.

Types of Styles and Style Sheets

There are three different ways in which we can associate styles to the elements of a particular web page:

using an external style sheet

By placing your **style rules** in an external style sheet, you can link this one file to any web pages on which you want those styles to be used. This makes updating a web site's overall look a cakewalk.

To reference an external style sheet from a web form, insert the following markup inside the `head` element:

```
<link rel="stylesheet" type="text/css" href="file.css" />
```

In the above example, `file.css` would be a text file containing CSS rules, much like the example shown below:

```
a
{
  background: #ff9;
  color: #00f;
  text-decoration: underline;
}
```

using an embedded style sheet

You can place style rules for a page within `<style type="text/css">` tags inside that page's `head`.

```
<style type="text/css">
  a
  {
    background: #ff9;
    color: #00f;
    text-decoration: underline;
  }
</style>
```

The problem with using these "embedded" styles is that we can't reuse those styles in another page without having to type them in again, which makes global changes to the site very difficult to manage.

using inline style rules

Inline styles allow us to set styles for a single element using the `style` attribute. For instance, we might give a paragraph a border, and color it red, with the following markup:

```
<p style="border-style: groove; color: red;">
  Copyright 2006
</p>
```

When used in embedded or external style sheets, the first part of any style rule must determine the elements to which the rule will apply; we do this using a **selector**. In ASP.NET, we typically use two types of selectors:

element type selectors

An element type selector targets every single instance of the specified element. For example, if we wanted to change the colour of all level two headers in a document, we'd use an element type selector to target all `<h2>`s:

```
h2
{
  color: #369;
}
```

classes

Arguably the most popular way to use styles within your pages is to give each element a `class` attribute, then target elements that have a certain `class` value. For example, the following markup shows a paragraph whose `class` attribute is set to `fineprint`:

```
<p class="fineprint">
  Copyright 2006
</p>
```

Now, given that anything with the class `fineprint` should be displayed in, well, fine print, we can create a style rule that will reduce the size of the text in this paragraph, and any other element with the attribute `class="fineprint"`:

```
.fineprint
{
  font-family: Arial;
  font-size: x-small;
}
```

Whether you're building external style sheets, embedded style sheets, or inline style rules, style declarations use the same syntax.

Now that you have a basic understanding of some of the fundamental concepts behind CSS, let's look at the different types of styles that can be used within our ASP.NET applications.

Style Properties

You can modify many different types of properties using style sheets. Here's a list of the most common property types:

font
: This category provides you with the ability to format text level elements, including their font faces, sizes, decorations, weights, colors, etc.

background
: This category allows you to customize backgrounds for objects and text. These values give you control over the background, including whether you'd like to use a color or an image for the background, and whether or not you want to repeat a background image.

block
: This category allows you to modify the spacing between paragraphs, between lines of text, between words, and between letters.

box
: The box category allows us to customize tables. If you need to modify borders, padding, spacing, and colors on a table, row, or cell, use the elements within this category.

border
: This category lets you draw boxes of different colors, styles, and thicknesses around page elements.

list
: This category allows you to customize the way ordered and unordered lists are created.

positioning
: Modifying positioning allows you to move and position tags and controls freely.

These categories provide an outline of the aspects of a design that can typically be modified using CSS. As we progress through the book, the many types of style properties will become evident.

The `CssClass` Property

Once you've defined a class in a style sheet (be it external or internal), you'll want to begin to associate that class with elements in your Web Forms. You can associate classes with ASP.NET Web server controls using the `CssClass` property. In most cases, the value you give the `CssClass` property will be used as the value of the resulting element's `class` attribute.

Let's see an example. First, create in your `Learning` folder a file named `Styles.css`, and copy this code into it:

File: **`Styles.css`**

```
.title
{
  font-family: Arial, Helvetica, sans-serif;
  font-size: 19px
}
.dropdownmenu
{
  font-family: Arial;
  background-color: #0099FF;
}
```

```
.textbox
{
  font-family: Arial;
  background-color: #0099FF;
  border: 1px solid
}
.button
{
  font-family: Arial;
  background-color: #0099FF;
  border: 1px solid
}
```

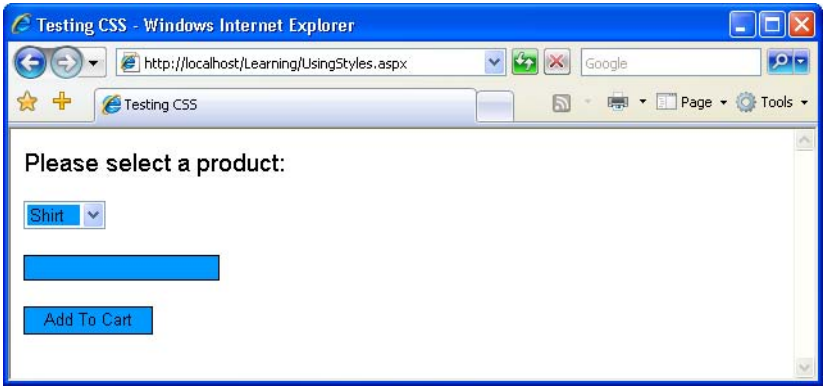
Then, create a new file named `UsingStyles.aspx` with this code:

File: **UsingStyles.aspx (excerpt)**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>Testing CSS</title>
    <link href="Styles.css" type="text/css" rel="stylesheet" />
  </head>
  <body>
    <form runat="server">
      <p class="title">Please select a product:</p>
      <p>
        <asp:DropDownList id="productsList"
            CssClass="dropdownmenu" runat="server">
          <asp:ListItem Text="Shirt" selected="true" />
          <asp:ListItem Text="Hat" />
          <asp:Listitem Text="Pants" />
          <asp:ListItem Text="Socks" />
        </asp:DropDownList>
      </p>
      <p>
        <asp:TextBox id="quantityTextBox" CssClass="textbox"
            runat="server" />
      </p>
      <p>
        <asp:Button id="addToCartButton" CssClass="button"
            Text="Add To Cart" runat="server"  />
      </p>
    </form>
  </body>
</html>
```

Loading this page should produce the output shown in Figure 4.13.

Figure 4.13. CSS at work



In the next chapter, we'll learn to use Visual Web Developer to create CSS definitions through a simple visual interface.

Summary

In this chapter, we discussed web forms, HTML server controls, web server controls, web user controls, master pages, and CSS. All these elements can be combined to create powerful structures for your web sites.

In the next chapter, we'll start building "real" web applications, putting into practice most of the theory you've learned so far, and using a professional development environment that will do part of the work for you.

What's Next?

If you've enjoyed these chapters from *Build Your Own ASP.NET 2.0 Web Site Using C# & VB*, why not order yourself a copy? In the rest of the book you'll learn how to put ASP.NET to full use in a number of practical solutions including discovering how to use databases in conjunction with ASP.NET to create dynamic web applications.

This second edition of this bestselling ASP.NET book has been completely revised for ASP.NET 2.0, with greatly expanded coverage of best practice techniques like code-behind, usage of Visual Web Developer and SQL Server Express Edition, and coverage of powerful new ASP.NET 2.0 functionality, including the GridView control and Master Pages.

Here are just a few of the things you'll learn to do:

- Write best practice code and use code-behind.
- Build a complete web application: the Dorknozzle intranet.
- Validate input using the Validation Controls.
- Learn database design and SQL using SQL Server Express Edition.
- Manage page content using the GridView and DetailsView controls.
- Secure your applications with Memberships and Roles.
- Work with files and email.
- And a whole lot more…

Purchase this book and gain access to the code archive, so you can try out all the examples without retyping. Just like in the book, all examples in the code archive are offered in C# and VB varieties.

Index

A

ABS function, 314
access modifiers, 82
AccessDataSource object, 470
accessing directories and directory information, 572, 583–586
accessors, 129–130
Active Server Pages (ASP) technology, 2
ActiveViewIndex property, 124–125
ActiveX Data Objects .NET (*see* ADO.NET)
Ad elements, 119
Add method, 346
Add New Column..., 442, 456
Add New Diagram, 280
Add New Employee button, 486–487
adding a new item to a project, 187–188
addition operator (+), 311
address book, 145, 429–435
  adding a new employee, 483, 485, 487
  code for, 488–489
  displaying details about the selected row, 445
  displaying selected columns, 437
  highlighting selected fields, 443
  in Design View, 430
  in DetailsView, 446, 448–450, 458
  in GridView, 433
  paging, 478
  SqlDataSource object code, 476
  styled, 440
  updating an employee's address and city, 467
  viewing employee details, 451
  with SqlDataSource control, 473
Address field, 462
admin newsletter, 146
admin tools, 146
Admin Tools link, 564
Admin Tools page, 379–381
  in Design View, 381
Administer Website link, 564
Administration Tool (*see* ASP.NET Web Site Administration Tool)
Administrative Tools folder, 6, 8
Administrators role
  creating users, 556
ADO.NET, 332–333
  built in protection against injection attacks, 528
  bulletproofing data access code, 351–354
  data source controls, 470–494
  defining the database connection, 334–336
  establishing link between database and application, 333
  executing the Command, 337–339
  importing the SqlClient namespace, 333–334
  preparing the command, 336
  reading the data, 342–343
  setting up database authentication, 339–342
  using parameters with queries, 344–350
  using the repeater control, 354–359
AdRotator control, 117–120, 613
  benefits, 117
Advanced SQL Generation options, 480
Advanced... button, 480
aggregate functions, 319–322
allowing user access, 540
AllowPaging, 477, 504

AllowSorting property, 477, 510
ALTER PROCEDURE, 329
AlternateText, 119
AlternatingItemStyle, 425
<AlternatingItemTemplate> template, 356, 405
AND operator, 313
anonymous users, 540, 559–560, 567
App_Code, 163
App_Data, 149, 163
App_Themes, 163
appending text, 579
AppendText method, 578, 580
Application Directory field, 20–21
application domain, 162
application level security policies, 163
Application object, 173
Application Settings, 162
application stabilization, 162
application state, 173–180
  difference from session state, 173
  locking, 178
  objects in, 173–174
  removing objects from, 173
  two users updating simuntaneously, 179
  two users updating with locks, 179
  using, 174–177
Application_AuthenticateRequest, 171
Application_AuthorizeRequest, 171
Application_BeginRequest, 171
Application_End, 171
Application_EndRequest, 171
Application_Error, 171
Application_PreSendRequestContent, 171
Application_PreSendRequestHeaders, 171
Application_Start, 171
arithmetic functions, 314–315
arrays, 62–65
  and VB, 207
  declaring, 63
  reading an element from, 63
ASP.NET, 1–2
  access to functionality of .NET Framework, 4
  advantages, 4
  and client-side validation, 222
  as server-side technology, 2, 29
  as technology for developing web applications, 3
  controls in, 95
  reuse of common User Interface elements, 4
  separation of server-side code from HTML layout, 4
  use of Microsoft .NET Framework, 4
  what is it?, 2–5
ASP.NET 2.0 (*see* ASP.NET)
ASP.NET code
  adding time information, 27
ASP.NET configuration sections, 169
ASP.NET events, 51
  outputs, 59
ASP.NET files
  access to, 13
ASP.NET function
  executing, 66
ASP.NET languages, 48
ASP.NET login controls (*see* login controls)
ASP.NET membership system, 544
  creating membership data structures, 544–547
  using your database to store membership data, 547–552
ASP.NET operators, 69
ASP.NET option, 20
ASP.NET page, 34
  as plain HTML, 32
  ASP.NET server controls, 40

code declaration blocks, 37–39
code render blocks, 39–40
compiling of, 4
directives, 36, 47
elements, 34
encryption of view state data, 46
HTML tags, 42–43
life cycle, 35
literal text, 42–43
parts of, 35
sample page, 36
server-side comments, 41–42
writing your first page, 26–32
ASP.NET page strcture, 34–43
ASP.NET runtime, 34
database connection error, 340
ASP.NET security, 530–544
ASP.NET server controls, 34, 40
advantages for ASP.NET developers, 40
residence of, 40
view state, 40, 44–47
ASP.NET support web site, 32
ASP.NET validation controls, 220–223
enforcing validation on the server, 223–228
reserved space for, 228
using, 229–242
ASP.NET Web Site Administration Tool, 545, 560
Application tab, 553
Provider tab, 553
Security tab, 545–546, 552
assigning the Users role, 558
Create user, 556
creating access rules, 559–560
creating the administration account, 556
Enable Roles, 554
Select authentication type link, 553
to create an application setting, 554
using, 552–554
ASP.NET Web Site template, 147
asp: prefix, 101–102
<asp:Label/> tag, 27, 29–31
ASPNET account
adding, 575
giving write access to, 575
aspnet_regsql.exe, 548
customization, 550
using at the command prompt, 550
ASPNETDB database, 545, 547
access to, 546
as User Instance database, 546
important features, 547
ASPNETDB database files, 547
AspNetSqlMembershipProvider, 556
overriding default settings, 557
.aspx files, 34, 47
Attachment class, 594
AttachmentCollection class, 594
Attachments (email messages), 594
authenticating users, 534, 563–564
authentication, 530
forms, 531–544
Passport, 531
Windows, 531
authentication mode
attribute values, 533
authentication ticket, 532, 537
authorization, 530
authorization section
configuring, 533
Auto Format, 424
AutoFormat..., 437
AutoGenerateColumns, 435
AVG function, 322

B

background (style property), 138
backslash (\) character
used inside a string, 336

banner advertisements, 117
  example, 118–120
Bcc (email messages), 594
BETWEEN keyword, 303–304
BindData method, 506
BindGrid method, 430, 500
  modifying, 507–508
  to add a new department to a database, 522–523
  to apply sorting, 513–515
BindList method, 410
bit data type, 263
block, 138
Body (email message), 594
bool (C#), 60
Boolean (VB), 60
border, 139
BoundField column, 436, 441, 446, 461
  converted to TemplateFields, 463
box, 139
breadcrumb navigation system, 122
break (C#), 76
breaking long lines of code, 70
breakpoint, 208
Browse With, 160
Build Style..., 191
BulletedList control, 112, 614
BulletStyle property, 112
Button control, 105–106, 224, 345, 347, 602
  attributes, 52, 54
  events, 615
  OnClick attribute, 536
  properties, 614
button_Click routine, 53
ButtonField column, 441–443

C

C#, 4, 28, 48–49
  arrays, 63
  as strongly-typed language, 61
  case sensitivity, 29
  case sensitivty, 71
  Click event, 53
  code-behind files, 89
  comments in, 37
  curly braces to mark end of script, 30–31
  data types, 60
  declaring an array, 64
  do while loop, 74
  editing Default.aspx, 152
  enabling Debug Mode, 167
  equality operator, 69
  file upload, 592
  for loop, 75
  functions, 65
  HTML server controls in, 100
  if else statement, 70
  operators, 69
  page events, 57
  <script> tags in, 28, 37
  selecting case, 71
  semicolon to end lines of code, 30
  semicolon to mark end of a command, 70
  simple button and label (code-behind files), 89
  simple button and label (non code-behind files), 88
  square brackets for arrays, 64
  static method, 578
  subroutine components, 55
  while loop, 73
C++, 48
Cache object, 182–183
calendar
  default, 113
Calendar control, 112–116, 135, 192
  events, 617
  properties, 114–115, 615–617

user selection of days, weeks and months example, 115–116
call stack, 214
Camel Casing, 102
Cancel button, 459
Cancel Editing button, 416–417
candidate key, 266
Cascading Style Sheets (CSS), 2, 135–136, 189–190, 193
template styles, 438
Case, 585
case sensitivity, 71
C#, 29
selecting case, 71
Cassini, 5, 12, 156
configuration options, 20
configuring, 12
installing, 9
permissions, 573
using, 20–21
cast, 61
Catch block, 214–217, 354, 364, 390
error message, 373
Category drop-down list, 365
CausesValidation, 226
CC (email messages), 594
CEILING function, 314
ChangeExtension method, 589
ChangeMode method, 459
ChangePassword control, 562
char (C#), 60
Char (VB), 60
CHARINDEX function, 316–317
CheckBox control, 107, 111, 617
CheckBoxField column, 441
CheckBoxList control, 111, 617
CheckChanged event, 107
CheckUniqueUserName method, 242
child tag, 355
CircleHotSpot control, 108
City field, 462
class attribute (style), 137
classes, 77, 81
access modifiers, 82
definition, 78
for working with directories, 572
for working with files, 572
for working with streams, 572
importing namespaces into, 86
instantiating, 81
methods, 79, 81
.NET Framework, 30
properties, 79
scope, 82
Click event, 52–53, 105–106, 615
Click event handler, 100, 223, 225, 248, 345, 347, 372, 386
Click subroutine, 91
client-side technologies, 2
client-side validation, 219–220
advantages/disadvantages, 219
and ASP.NET 2.0, 222
of user input, 528
ClientValidationFunction proprety, 242
code-behind files, 87–91, 129, 152
partial classes, 90–91
separation of presentational elements (controls) and your code, 87
code declaration blocks, 29, 34, 37–39
code within, 39
comments in VB and C# code, 37–39
language attribute, 38–39
placement on ASP.NET page, 38
src attribute, 39
Code Editor, 151–153
code isolation, 162
code render blocks, 34, 39–40, 87
types of, 39
collections, 173
colour setting, 156
column controls, 436, 441
column headers, 435–436
columns, 253, 297

choosing, 475
customizing in GridView, 435–436
displaying selected, 437
properties, 264–265
read-only, 461
Combine method, 589
combining lines of code, 70
CommandField column, 441, 456
company newsletter page
creating, 601–610
CompareValidator control, 231–233, 628
difference from RequiredFieldValidator control, 232
example, 231
for data type checks, 233
to compare value of control to a fixed value, 232
values, 232
compilation errors, 210
computer name, 341
conditional logic, 70–71
configuration errors, 210
configuration file
elements, 169
configuration section groups, 169
configuration section handler declarations, 170
configuration sections, 169
configuration settings, 169
ConfigurationManager class, 364, 554
Configure Data Source, 473, 479, 490
configuring
Cassini, 12
Internet Information Service, 11
web server, 11–21
confirmPasswordTextBox control, 230
connection string, 334, 336, 363, 578
in Web.config, 364
specifying, 474, 479
constraints, 266
constructors, 81
Content control, 134
ContentLength, 590
ContentPlaceHolder, 200
ContentPlaceHolder control, 133–135
ContentType, 591
control binding, 365
Control class, 85
control events, 52–56
subroutines, 54–56
ControlToCompare property, 232
ControlToValidate property, 230, 232
Convert this field into a TemplateField link, 460
Convert to Template option, 564
cookieless attribute, 538
cookies, 183–186
creating, 185
COUNT function, 319
CREATE PROCEDURE, 327, 329
CreateText method, 578, 580
CreateUserWizard control, 562
creating users and roles, 554–556
<credentials> tag, 541–542
CSS (*see* Cascading Style Sheets)
CssClass property, 139–141
current databases, 256
custom errors, 212–213
Custom Errors option, 20
customErrors, 212
CustomValidator control, 239–242, 629

D

data access code
bulletproofing, 351–354
data adapters, 497
data binding, 355, 365–371, 410
and sorting, 516
DataSet to GridView control, 509
DefaultView does not apply when binding to a DataSet, 510

DetailsView to a SqlDataSource, 479–489
for creating drop-down lists, 366
GridView to a SqlDataSource, 472–478
data consumer, 365
data key, 446
data readers, 494, 496
disadvantages, 494
retrieving data using, 495
data sets, 494
advantages over data readers, 495
breaking ties to data source, 495
component parts, 497–498
memory and resource requirements, 496
shared among multiple requests, 496
storing in View State, 506–509
data source, 365
testing, 476
data source controls, 470–494
binding the DetailsView to SqlDataSource, 479–489
binding the GridView to a SqlDataSource, 472–478
displaying lists in DetailsView, 489–492
in the Visual Web Developer's Toolbox, 471
data store
customizing, 548
Data tab, 471–472
data tables
in ADO.NET 2.0, 498
populating, 273–275
data types, 59–60, 262–264
aliases, 60
avoid mixing of, 61
converting from one type to another, 61
database
adding stored procedures, 397–399
column properties, 264–265
creating, 254–257
using SQL Server Management Studio, 256–257
using Visual Web Developer, 255
deleting records, 394–397
inserting records, 371–378
primary keys, 265–267
updating from a modified DataSet, 521–526
updating records, 378–393
what is it?, 252–254
database authentication
setting up, 339–342
database connection
defining, 334–336
database connection error, 340
database design, 258–259
and implementation, 260
delete anomaly, 259
logical design, 258, 261
overcoming problems, 259
physical design, 258
relational database concepts, 276–292
update anomaly, 259
database diagrams, 280–284
adding support for, 281
adding tables to, 282
and table relationships, 287–292
to create foreign keys, 280
to provide visual representation of tables in the database, 280
visualizing data tables using, 283
zooming feature, 282
Database Diagrams node, 280
Database Engine Query, 295
Database Explorer, 144, 267, 270
Database field, 548
Database name field, 257
database normalization, 262
database protection, 528

database server, 252
database tables, 253
  creating, 258–262, 271–273
  for storing lists of categories and subjects, 365
  inserting data and identity columns, 273
  populating, 273–275
  relationships between tables, 260, 287–292
Databases node, 256–257
DataBind method, 585
DataColumn class, 497
DataField property, 436
DataGrid, 428
DataGrid control, 354
DataKeyNames property, 447
DataList control, 354, 401, 427
  and Visual Web Developer, 422–424
  basics, 402–405
  generation of ItemCommand event, 408
  handling buttons within their templates, 406, 408, 411
  smart tag, 422
  style elements, 425
  styling, 424–426
  versus Repeater control, 404
DataList events, 406–413
  using controls within, 412
DataList items
  editing, 413–422
DataList templates, 405
  building visually, 423
DataRelation class, 497
DataRow class, 497
DataSet class, 470, 493, 497
DataSet mode, 493
DataSet object, 493–494, 502
DataSetS
  adding DataTable to, 502
  binding to controls, 498–504
  difference from databases, 498
  filling with data, 502
  for updating a database, 521–526
  structure, 498
Datasheet View, 273
DataSource, 585
DataSourceMode, 493
DataTable class, 470, 497
DataTable object, 493
  in ADO.NET 2.0, 498
DataTableS, 504
  displaying data from in a GridView, 503
DataTypeCheck, 233
DataView class, 470, 497, 521
DataView object, 493, 510
  sorting parameters, 510
DATEADD function, 318
DATEDIFF function, 318
date-of-birth text box, 233–234
DATEPART function, 318
dates
  specifying, 234
DateTime class, 30
  Now property, 30
    ToString method, 30
datetime data type, 263
DAY function, 318
dbErrorLabel control, 382
Debug toolbar, 210
debugging, 158, 200
  run-time error, 206
  setting using Web.config, 167
  stopping, 160
  using a breakpoint in the code, 208
  with Visual Web Developer, 204–210
decimal (C#), 60
Decimal (VB), 59–60
declaring a function, 66–67
declaring an array, 63
declaring an object, 78

default event handlers, 444
default names
  adding to a document, 164
DEFAULT property (columns), 264
default validation groups, 245
default web browsers, 159
default web form, 149
Default Web Site, 13, 15
Default.aspx, 149–150
  deleting, 199
  editing, 152
  viewing, 151
DefaultView
  does not apply when binding to a DataSet, 510
Delete Employee button, 395
DELETE query, 394, 480
DELETE statement, 325–326
DeleteCommand property, 521
deleting database records, 394–397
deleting records, 326
denying user access, 539–541, 559
Department drop-down list, 490
Department ID TemplateField, 491
Departments, 145, 515
  adding many new to a database, 524
  data source, 491
  deleting, 524–526
  viewing with paging functionality, 506
Departments page, 498–502
  storing in View State, 506–509
Departments table (employee database), 271, 274, 298
  primary key, 279
  retrieving department names and their ID, 300
Design button, 150, 198
Design View, 151, 200, 361, 381, 386, 430, 442
DetailsView action types
  and the events they trigger when clicked, 454
DetailsView control, 354, 470
  adding a new employee, 485
  binding to SqlDataSource, 479–489
  deleting properties from, 472
  display modes, 458
  displaying lists, 489–492
  Edit button, 452
  entering edit mode, 456–459
  in edit mode, 453, 458
  properties to set, 482
  recreating the columns, 482
  skins in, 451
  styling, 450–452
  synchronization with GridView control, 486
  updates and GridView control, 485
  use with address book, 446, 448–450
  using, 445–450
  using templates, 459–463
DetailsView events, 452–456
DetailsView object
  storing ID of the record, 466
DetailsView records
  updating, 463–468
  using a stored procedure, 467
development languages, 48
Dim (dimension), 59
directives, 34, 36
  location on page, 37
  types of, 36, 47
  working with, 47
directories, 584
  working with, 586–589
directories and directory information
  accessing, 572, 583–586
Directories/Files, 584
Directory Browsing, 18
Directory class, 583
Directory Security option, 19

DirectoryInfo class, 583
disabling
  file sharing, 574
disconnected data, 494
disconnected data access, 493
display modes
  DetailsView control, 458
Display property, 228
Display View, 477
displaying data correctly, 529
displaying lists in DetailsView, 489–492
DisplayMode property, 112
disposal of objects, 577
DISTINCT clause, 300–302
division operator (/), 311
Do loop, 72
Do While loop, 74
Documents option, 19
dog
  OOP example, 78–81
Dorknozzle project, 144–146
  adding a foreign key, 285
  adding a skin, 193
  adding an employee help desk request web form, 201–204
  adding files, 187
  adding many new departments, 524
  address book, 429–433
    adding a new employee, 483, 485, 487
    displaying details about selected rows, 445
    displaying selected columns, 437
    highlighting selected fields, 443
    paging, 478
    showing DetailsView control, 447
    updating employee's address and city, 467
    viewing employee details, 451
  applying the themes, 194
  building the master page, 195–198
  creating a company newsletter page, 601–610
  creating a new style sheet, 190–192
  creating a new theme folder, 189
  creating employee directory, 360–399
  creating the employees table, 267–270
  creating the remaining tables, 271–273
  creating virtual directory using IIS, 161
  database diagrams, 280–284
  deleting records, 394–397
  Departments, 515
    data source, 491
    viewing with paging functionality, 506
  Departments page, 502
  employee records database, 252, 256, 258, 261
  executing stored procedure, 328–330
  extracting information from employees database, 294
  implementing relationships in employee database, 284–287
  inserting records, 371–378
  login page, 534, 565
  populating the employee records database, 273–275
  preparing the sitemap, 187–189
  relationships in employee records database, 286
  running through IIS, 341
  running through Visual Web Developer's integrated web server, 339
  starting, 186–200
  styled address book, 440
  styling web server controls, 192–193
  updating Help Desk page, 245–250
  updating records, 378–393

using stored procedures, 397–399
using the master page, 199–200
using validation controls, 245–250
view the logged-in user sees, 565
web application, 148
welcome page, 201
Dorknozzle Properties window, 161, 164
drop-down list
created with data binding, 366
selecting directories or files from, 584
DropDownList control, 101, 110, 203, 366, 386, 584, 619
Dynamic mode (validation), 228

E

e As EventArgs (VB), 56
Edit button, 452, 454, 459
Edit columns, 459
Edit employee button, 414
Edit fields, 459
Edit Fields, 491
Edit link, 457
edit mode
DetailsView control, 453, 456–459
GridView control, 453, 456–459
Edit Templates, 423
editing
DataList items, 413–422
field's properties, 460
EditItemStyle, 426
<EditItemTemplate> template, 405, 414, 461
EditRoleGroups, 568
EditUpdate checkbox, 456
element type selectors, 137
Else statement, 70
email
configuring the SMTP server, 595–597
sending a test email, 597–600
sending with ASP.NET, 593–610
email address
invalid, 237
embedded style sheets, 137
employee database, 253, 258, 266
creating the employee table, 267–270
creating the remaining tables, 271–273
entities, 261
relational design concepts, 276–278
with Department ID field, 260
employee details, 387
employee directory, 145, 182, 404
completed page, 361
creating, 360–399
data binding, 365–371
deleting records, 394–397
hiding employee details, 407
inserting records, 371–378
showing employee ID, 411
styled list, 425
updated using DataList, 403–404
updating records, 378–393
using stored procedures, 397–399
viewing an employee in edit mode, 421
employee help desk request web form, 201–204
employee ID, 344–345, 349, 376, 386, 411
invalid, 351
employee list, 343
in a drop-down, 386
employee table, 274
creating, 267–270
extracting information from, 294
referencing records from the Departments table, 279
structure, 270
using Repeater control, 356–360

employeeDataSource, 488
employeesList control, 382
  populating with list of employees from database, 382–385
encryption, 529
  asymmetric algorithms, 529
  symmetric algorithms, 529
End Sub, 30–31
Enforce Foreign Key Constraint, 290
entities (tables), 258
Equals, 85
error messages, 205–206, 210, 212–213
  invalid email address, 237
  validation errors, 236
Event button, 463
event handler, 52, 58
event receiver, 83
event sender, 83
EventArgs e (C#), 56
events, 51
  (*see also* control events; page events)
  naming using past and present tense, 454
  triggered by DetailsView action types, 454
  triggered by GridView action types, 454
events (object), 77, 83
"everything is an object", 84
Exception class
  properties, 217
Exception object, 217
exceptions, 206
  handling locally, 213–218
Execute button, 296
ExecuteNonQuery method, 337, 371, 376
ExecuteReader method, 337, 339
ExecuteScalar method, 337
executing a page, 158
  with debugging, 158
  without debugging, 158
Exit (VB), 76
exiting loops prematurely, 76
expressions, 310
expressions (SQL), 310
external style sheets, 136

F

fields
  choosing, 479
fields (database), 253
fields (object), 77
field's properties
  editing, 460
File class, 572–573
file paths
  working with, 586–589
file sharing
  disabling, 574
FileBytes property, 590–591
FileContent property, 590–591
FileName property, 590
files, 584
  uploading, 590–593
Filestream class, 572
FileUpload control, 125, 590–591, 619
Fill method, 502
filtering data, 520
filtering groups, 321–322
Finally block, 214–215, 351, 364
FindControl method, 412, 466
FirstBulletNumber property, 112
float data type, 263–264
floating point numbers, 59
FLOOR function, 314
font, 138
FooterStyle, 426
<FooterTemplate> template, 356, 405
For Each loop, 72, 75
For loops, 72, 75–76, 205
For Next loop, 72
foreign keys, 278–280

adding to Dorknozzle database, 285
creating using database diagrams, 280
disallowing the addition of invalid data, 286
editing, 285
options, 290
properties, 289
<form runat="server"> tag, 94, 102
forms authentication, 531
as cookie-based, 532
authenticating users, 534
configuring, 538–539
constructing a basic login page, 532–534
working with, 532–544
working with hard-coded user accounts, 535–538
forms authorization
configuring, 539–541
FormsAuthentication class, 532, 537
Authenticate method, 542
HashForStoringInConfigFile method, 543
SignOut method, 544
FormsAuthenticationTicket class, 532
FormsIdentity class, 532
FormView control, 354
From (address in email message), 594
FROM clause (SELECT query), 298
fully qualified name, 86
functions, 65–68
and return types, 66–67
declaring, 66–67
number of parameters, 67

G

generic Control objects, 412
Get accessor, 130
GETDATE function, 317
GetDirectories method, 583, 585–586
GetDirectoryName method, 588–589
GetExtension method, 588–589
GetFileName method, 588–589
GetFileNameWithoutExtension method, 588–589
GetFiles method, 583, 585–586
GetFileSystemEntries method, 583, 585–586
GetFullPath method, 589
GetHashCode, 85
GetPathRoot method, 589
GetTempFileName method, 589
GetTempPath method, 589
getting help, 32
GetType, 85
Global Application Class template, 172
Global.asax, 163, 170–173
changing, 173
creating, 172
event handler in, 172
Timeout property in, 181
greater-than operator (>), 311
greater-than or equal-to operator (>=), 312
grid records
adding a new field, 442
selecting, 440–445
gridSortDirection property, 516–517
gridSortExpression property, 516–517, 520
GridView action types
and the events they trigger when clicked, 454
GridView control, 354, 470, 584–585
and DetailsView update, 485
automatic presentation features, 435
avoid overwriting columns, 477
binding to DataSet, 509
binding to SqlDataAdapter and a DataSet, 500
binding to SqlDataReader, 429
binding to SqlDataSource, 472–478

calling BlindData method, 506
column controls, 436, 441
customizing columns, 435–436
data key feature, 446–448
deleting properties from, 472
differences from Repeater control, 429
displaying data drom a DataTable, 503
entering edit mode, 456–459
features, 428
in edit mode, 453, 456
PageSize, 477, 504
paging, 478, 504–506
proprties to set, 477
selecting grid records, 440–445
skins in, 439
smart tag options, 437
Sorting property, 510–512
styling with templates, skins and CSS, 436–440
synchronization with DetailsView control, 486
table generation, 434
template styles, 437
using, 428–435
using templates, 459–463
versus DataGrid, 428
GridView events, 452–456
GROUP BY clause, 319–321
GroupName property, 107
groups of values, 318–322
COUNT function, 319
filtering groups using HAVING, 321–322
grouping records using GROUP BY, 319–321

H

Handles keyword, 249
hard-coded user accounts, 535–538
HasExtension method, 589
HasFile property, 590
hashing, 529
passwords, 542–543
hashing algorithm, 542
HAVING clause, 321–322
HeaderStyle, 425
<HeaderTemplate> template, 355, 405
HeaderText property, 436
help desk, 145
help desk page, 201–204
updating, 245–250
Help Desk Request form, 377
help services, 32
HelpDesk table, 372, 378
setting the identity property, 376
HelpDesk table (employee database), 271, 276, 278
sample data from, 278
HelpDeskCategories table (employee database), 271, 275–277
HelpDeskStatus table (employee database), 272, 275–277
HelpDeskSubjects table (employee database), 272, 275–277
hidden form fields, 46
HiddenField control, 105, 620
HidePanel subroutine, 110
hierarchical structure of terms, 120–123
hit counter, 181
hit counters, 174–180
homepage, 145
Host property, 608
hot spots, 108–109
HotSpotMode property, 109
possible values, 108
HTML, 2, 31
<img> tag, 108
div element, 109
select element, 110–111
HTML code, 95
viewing, 31

HTML comments, 41
HTML control classes, 96
HTML documents, 607
HTML elements
  access to, 40
HTML hidden form field, 46
HTML output
  visitors' browser interpretation of, 529
HTML pages, 94
HTML server controls, 95–97, 643–658
  accessing properties of, 99–100
  assigning IDs t, 101
  essentially as HTML tags with run-at="server" attribute, 98
  survey form example, 97–101
  using, 97–101
HTML tags, 34, 42–43
  in HTML server controls, 98
  manipulation, 96
HtmlAnchor control, 644
HtmlButton control, 97–98, 109, 644
HtmlForm control, 97–98, 645
HtmlGeneric control, 646
htmlIinputImage control, 651
HtmlImage control, 647
HtmlInputButton control, 647
HtmlInputCheckBox control, 648
HtmlInputFile control, 649
HtmlInputHidden control, 650
HtmlInputRadioButton control, 652
HtmlInputText control, 97–98, 653
HtmlSelect control, 97–98, 653
HtmlTable control, 655
HtmlTableCell control, 656
HtmlTableRow control, 657
HtmlTextArea control, 658
HTTP Headers option, 19
HttpCookie class, 184
HTTPS (HTTP Secure) protocol, 529
HttpUtility.HtmlEncode, 529
HyperLink control, 107, 620
HyperLinkField column, 441

I

identity increment, 265
IDENTITY primary key, 329
IDENTITY property (columns), 265
  and primary key, 267
  what they are not for, 275
identity seed, 265
identity values, 324
IDENTITY_INSERT property, 267
If statement, 69, 72
  combined with Else statement, 70
  VB code, 71
if statement (C# code), 71
IIf (in VB), 484
IIS (*see* Internet Informaton Services)
Image control, 108, 621
ImageButton control, 106, 621
ImageField column, 441
ImageMap control, 108–109, 622
Images folder, 198
ImageURL, 119
ImageUrl attribute, 107
Import directive, 36, 47
Imports (VB), 86
Impressions value, 119
IN operator, 313
  use in SELECT queries, 305–306
IndexOutOfRangeException class, 206
inheritance, 83
initialization (variables), 59
inline code, 39
inline expressions, 39–40
inline style rules, 137
inner join, 309
input element, 101
inputString, 581
Insert method, 182
INSERT query, 371, 480
INSERT statement, 323–324, 329

syntax, 323
InsertCommand property, 521
inserting database records, 371–378
<InsertItemTemplate> template, 461
installing
Cassini, 9
Internet Information Services (IIS), 6–8
.NET Framework, 10
.NET Framework Software Development Kit, 10
required software, 5–6
SQL Server 2005 Express Edition, 22
SQL Server Management Studio Express, 22–24
Visual Web Developer 2005 Express Edition, 25
web server, 6–9
instance methods, 578
instantiating that class, 81
int (C#), 60, 67
int data type, 263
int.TryParse method (C#), 349
Int32, 480
Integer (VB), 59–60, 67
Integer.TryParse method (VB), 349
integers, 59, 61
converting to a string, 67
Integrated Windows Authenticartion, 339, 341
IntelliSense, 153–154
Internet Explorer, 159
Internet Explorer 6, 5
Internet Information Services (IIS), 5
amd Windows XP Home Edition, 156
and URLs, 158
configuring, 11
configuring SMTP server, 595
creating a new virtual directory, 161
creating web project in, 148
Default Web Site, 13, 15
Directory Browsing, 18
file placement, 12
installing, 6–8
loading errors, 14
permissions rule, 573
running Dorknozzle through, 341
running web applications in, 160–166
shortcut to, 6, 8
stopping and starting, 15
to connect to SQL server, 339, 341
using localhost, 13–15
using with Visual Web Developer, 165–166
virtual directories, 15–20
Internet Information Services (IIS) Administration Tool, 8
intranet functionality, 144
invalid input data error message , 349
IP address
setting, 597
Is Nothing, 177
IsBodyHtml property, 594, 607
IsPathRooted method, 589
IsPostBack, 369
using appropriately, 369
ItemCommand, 454
ItemCommand event handler, 408–409, 412
ItemDeleted event, 485
ItemInserted event, 485
ItemStyle, 425
<ItemTemplate> template, 355, 404–405, 413, 461
Button control in, 406, 408
ItemUpdated event, 485

J

J#, 49
Java, 49

JavaScript, 2, 219
diabling in Opera, 224
disabling in Firefox, 224
disabling in Internet Explorer, 225

K

keys, 265
Keyword, 119
KeywordFilter property, 119

L

Label control, 98, 100, 103–104, 174, 342, 580–581, 587–588, 602, 622
documentation, 103
Text property, 104
Label web server control, 101
LabelText property, 128–129
language attribute, 37–39
languages, 48
Lanugage drop-down list, 195
LEN function, 316
lenght of string function, 316
less-than operator (<), 312
less-than ot equal-to operator (<=), 312
LIKE keyword, 304–305
LIKE operator, 313
LinikButton control, 623
LinkButton, 408, 411
LinkButton control, 106
list (style property), 139
list controls, 110–112
ListBox control, 101, 111, 623
Literal control, 104, 408, 412, 624
setting its contents, 412
Text property, 104
literal text, 34, 42–43
LoadEmployeesList, 393
localhost, 13–15
Locals window, 208
LocalSqlServer, 547
connection to a disconnected database, 551
connection to your database, 551
definition in machine.config, 551
Location drop-down, 148
lock application state, 178
Log In button, 244
logged-in user
checking status of, 568
LoggedInTemplate, 567
logging users out, 543
logical design (database), 258, 261
Login control, 562
login controls, 544, 561–562
authenticating users, 563–564
customizing user display, 564–569
options for, 564
using, 563
login credentials, 541
validation, 536
login page, 565
configuring the authentication mode, 532
configuring the authorizatoin section, 533
construction, 532–534
creating the, 534–535
example, 537
for hard-coded user accounts, 535–537
naming, 534
login web form
and validation groups, 242
checking uniqueness of login, 240
creating, 221
list of error messages, 235
username field, 237
validating data in password fields, 231
LoginName control, 562, 567
LoginStatus control, 562, 567
loginUr1 attribute, 538

LoginValidationGrouo, 244
LoginView control, 562, 567
Logout link, 567
loops, 72–76
LOWER function, 315
lowercase, 315
LTRIM function, 315

M

machine name, 341
machine.config file, 551, 556
  overriding settings, 167
MailAddress class, 594
MailAddressCollection class, 594–595
MailMessage class, 593, 606
  proerties supported, 594
many-to-many relationships, 290–292
MapPath method, 579–580
mapping tables, 291
master page
  building, 195–198
  editing a web form that uses a, 201
  using, 199–200
master page files, 133
Master Page template, 195
master pages, 95, 132–135, 566
  differences from web forms, 134
  example, 133–134
  using, 135
MAX function, 322
MaximumValue poperty, 234
membership data
  storage using your database, 547–552
membership data structures, 544–547
membership system, 544
  login controls, 544
membership tables, 552
Menu control, 123, 197
  events, 639
  methods, 639
  properties, 635–639
Message property, 217
messageLabel object, 91
methods (class), 79, 81
methods (object), 77
Microsoft, 2
Microsoft .NET Framework (*see* .Net Framework)
Microsoft MSDN 2005 Express Edition, 26
Microsoft Passport accounts, 531
Microsoft SQL Server, 255
Microsoft SQL Server 2005 Express Edition (*see* SQL Server 2005 Express Edition)
Microsoft SQL Server Management Studio Express (*see* SQL Server Management Studio Express)
MIN function, 322
MinimumValue property, 234
MOD function, 315
ModeChanging event, 457
Modify, 270
money data type, 263–264
MONTH function, 318
mulitple tables
  querying
    using table joins, 309–310
multi-column keys, 266
multiple character operator (% or *), 313
multiple projects
  working with, 246
multiple tables
  querying, 309
    using subqueries, 308–309
  reading data from, 307–310
multiplication operator (*), 311
MultiView control, 123–125
  example, 124
Multiview control, 624

mySubName (...), 55

N

name attribute, 538
named skin, 193
namespace references, 168
namespaces, 86
  impiorting, 86
NavigateURL, 119
NavigateUrl property, 107
navigation web controls, 634–643
nchar (n) data type, 263
.NET
  objects in, 84–86
.NET Framework, 4–5
  availability, 5
  installing, 10
  languages supported, 4
.NET Framework 2.0 (*see* .NET Framework)
.NET Framework 2.0 SDK Documentation, 103, 109
.NET Framework 2.0 Software Development Kit (SDK) (*see* .NET Framework Software Development Kit)
.NET Framework Class Library, 30, 86
.NET Framework Software Development Kit (SDK), 6
  downloading, 11
  installing, 10
new condition
  creating, 480
New Connection... button, 473
new database
  creating using SQL Server Management Studio, 256–257
  creating using Visual Web Developer, 255
New Databases..., 257
New Query button, 272, 295, 341
newsletter page, 146
  creating, 601–610
"No file uploaded!" error message, 592
None mode (validation), 228
normalization (database), 262
not equal to operator (<> or !=), 312
NOT operator, 313
Notepad, 27
Nothing, 182
null, 177, 182
NULL property (columns), 264–265
  and default values, 265
numbers
  (*see also* integers)
  converting to strings, 68
nvarchar (n), 346
nvarchar (n) data type, 263

O

Object (data type), 60
Object class, 85
  public members, 85
Object Explorer, 270, 295
object IDs
  naming using Camel Casing, 102
object oriented programming, 76
  classes, 78–79, 81
  constructors, 81
  events, 83
  inheritance, 83
  methods, 81
  objects, 77–78
  properties, 80
  Rayne (dog) example, 78–81
  scope, 82–83
Object s (C#), 55
ObjectDataSource object, 470
objects
  behaviour, 77
  declaring, 78
  definition, 77

disposal, 577
events, 77, 83
fields, 77
in .NET, 84–86
methods, 77
properties, 77
state, 77
OnCheckChanged attribute, 107
OnClick attribute, 52–54, 105
OnClick property, 224, 249
OnCommand attribute, 54
OnDataBinding attribute, 54
OnDisposed attribute, 54
one-to-many relationships, 288–290
one-to-one relationship, 288
OnInit attribute, 54
OnItemUpdating property, 472
OnLoad attribute, 54
OnModeChanging property, 472
OnPreRender attribute, 54
OnSelectedIndexChanged property, 472
OnUnloaD attribute, 54
OOP (*see* object oriented programming)
Open Table Definition, 270
OpenText method, 580–581
operators, 68–70
definition, 68
to break long lines of code, 70
to combine lines of code, 70
operators (SQL), 311–313
OR operator, 312
ORDER BY clause
for sorting query results, 306–307
specifying, 475
ORDER BY... button, 474
out parameters, 349
overwriting text, 578

P

page
definition, 56
Page class, 95
documentation, 85
page counter, 181
page counters, 174–180
Page directive, 36, 47, 90, 134
page events, 56–58
order of execution, 57
subroutines, 56
page templates, 132–135
Page.IsValid property, 225–226, 242, 245
Page_Init event, 56
Page_Load event, 56
Page_Load method, 181, 335, 361, 366, 369, 410, 430
Page_PreRender event, 56
Page_UnLoad event, 57
PageIndex property, 505
PageIndexChanging event, 504
PageIndexChanging event handler, 505
PagerStyle, 477
pages element, 194–195
PageSize, 477
paging, 478, 504–506
paging buttons, 477
Panel control, 109–110, 123, 625
parameters, 346
in functions and subroutines, 67
out, 349
use with queries, 344–350
parent tag, 355
parser errors, 210
partial classes, 90–91
usage, 91
Pascal Casing, 101
Passport accounts, 531
Passport authentication, 531
password confirmation text box, 226

password strength requirements
changing, 556–558
PasswordRecovery control, 562
passwords, 536
default security, 556
encryption, 529
hashing, 542–543
verification, 537
passwordStrengthRegularExpression, 558
PasswordTextBox control, 227, 230, 232
path attribute, 538
Path class, 573, 586, 588–589
permissions
reading and writing text files, 573–575
persistent cookies, 184
physical design (database), 258
Place code in separate file checkbox, 196, 199, 202, 334, 360
PlaceHolder control, 109, 625
PolygonHotSpot control, 108
populating data tables, 273–275
positioning, 139
post back, 369
PostedFile property, 590
POWER function, 315
primary keys, 265–267, 278
and IDENTITY property, 267
as constraints, 266
setting columns as, 268
Private (access modifier), 82
Private (VB), 55
properties
defining, 513
properties (class), 79–80
properties (object), 77, 80
Properties dialog
options, 19
Properties window, 155–156, 289, 409, 444, 463, 477, 504
colour setting, 156
Property Builder, 424
Protected (access modifier), 83
protection attribute, 538
Public (access modifier), 82
public (C#), 55
Public (VB), 55
public interface, 83

Q

queries, 294
executing, 296
limiting the number of results with TOP, 307
matching patterns with LIKE, 304–305
multiple tables, 309
row filtering with WHERE, 302–303
SELECT statement, 297–299
selecting certain fields, 299–300
selecting ranges of values with BETWEEN, 303–304
selecting unique data with DISTINCT, 300–302
sorting results using ORDER BY, 306–307
table joins, 309–310
using parameters with, 344–350
using the IN operator, 305–306
query window, 296
question mark symbol ("?"), 533, 540

R

RadioButton control, 107, 625
RadioButtonList control, 108, 111, 626
RangeValidator control, 233–234, 630
Read mthod, 343
reading from text files, 571, 580–582
permissions, 573–575
ReadLine method, 580, 582

read-only
in columns, 461
ReadOnly modifier, 130
read-only properties, 130
ReadText method, 580
records (database), 253
RecrtangleHotSpot control, 108
Redirect attribute, 213
ReferenceEquals, 85
Refresh Folder, 166
Refresh folder, 198
Register button, 244
Register directive, 47, 131
attributes, 131
registration form, 242
regular expressions, 238
resources, 238
RegularExpressionValidator control, 236–237, 631
some regular expressions, 238
relational database design concepts, 276–278
Dorknozzle database, 284–287
foreign keys, 278–280
using database diagrams, 280–284
Remember Me checkbox, 538
remote servers, 548
Remove button, 163
Repeater control, 354–359, 361, 404, 427, 429
application, 354
differences from DataList control, 401
templates, 355
versus DataList, 404
REPLACE function, 315
RequiredFieldValidator control, 221, 223, 226, 230, 244, 632
difference from CompareValidator control, 232
Response object, 184
Response.Write (classic ASP), 40
reusable visual styles, 135
roles
creating, 554–556
root node, 122
RowCommand, 454
RowEdited event, 456
RowFilter property, 521
rows (database), 253
RTRIM function, 315
runat="server" attribute, 28–29, 37, 95, 98, 101–102
run-time errors, 206, 213

S

s As Object (VB), 55
SaveAs method, 591
scope, 82–83
<script runat="server"> tag, 115
<script> tags, 28–29, 37
runat="server" attribute, 37
SDK (*see* .NET Framework Software Development Kit)
SDK Command Prompt window, 548, 550
secure communication channels, 529
securing web applications, 559–561
security
common approaches to, 527
resources, 527
security guidelines, 528–530
database protection, 528
display data correctly, 529
encryption use, 529
hashing, 529
keeping sensitive data to yourself, 529
using secure communication channels, 529
validating user input, 528
Security tab, 545

security-related concepts
authentication, 530–531
authorization, 530
Select button, 386–390
Select Case (VB), 71
Select Case statements, 585
select element, 101
Select master page checkbox, 199, 202, 334, 360
SELECT query, 297–299, 371
FROM clause, 298
order of fields in a table, 299
selecting all columns using *, 299
selecting certain fields, 299–300
syntax, 297
using DISTINCT clause, 300–302
using the IN operator, 305–306
using WHERE clause, 302
selectButton control, 382
selectButton_Click method, 390
SelectCommand property, 502, 521
SelectedIndexChanged event, 111–112, 443–444, 452
SelectedIndexChanged event handler, 444, 448, 455
SelectedIndexChanged events, 584
SelectedItemStyle, 426
<SelectedItemTemplate> template, 405
SelectionChanged event, 115–116
SelectionMode attribute, 111
semicolon, 70
in C# code, 30
Send Email button, 600
Send Newsletter button, 604–606
sending a test email, 597–600
sending email with ASP.NET, 593–610
sensitive data, 529
transferring, 530
<SeparatorTemplate> template, 356, 404–405
seriving a subclass, 84
Server Explorer, 144
Server field, 548
Server Port field, 20
serverModel option box, 101
server-side button, 52
server-side comments, 34, 41–42
differences from HTML comments, 41
server-side HTML server control, 95
server-side technologies, 2, 29
server-side validation, 219, 223–228, 528, 536
failure of, 226
ServerValidate event, 242
Session object, 181, 183
changing from Application object, 181
session state, 173, 180–182
objects in, 181
testing, 181
Set accessor, 129–130
SetFocusOnError property, 223
settings configuration on a per-application basis, 163
shared method, 578
Show Table Data, 273
SIGN function, 315
Sign Out button, 544
SignOut method, 544
Simple Mail Transfer Protocol (*see* SMTP)
Simple style, 424
single table
reading data from, 294–307
Site Map, 188
SiteMapDataSource class, 120, 197
SiteMapDataSource control, 121–122
SiteMapDataSource object, 471
SiteMapPath control, 122–123
SiteMapPath controls, 634
SiteName set, 173

SitePoint Forums, 32
Skin File template, 193
SkinId property, 193
skins, 136, 193, 439, 451, 477, 501
  adding, 193
  definition, 193
slidingExpiration attribute, 539
Smart host field, 597
smart tag, 422, 424
  Auto Format, 424
  in SqlDataSource control, 473, 479
  of GridView, 437
SmartBox control, 128–129, 131
SMTP, 595
SMTP server
  configuring, 595–597
  installing, 595
  routing email to, 598
  starting, 596
SmtpClient class, 593
Software Development Kit (SDK) (*see* .NET Framework Software Development Kit)
Solution Explorer, 149–150, 160, 166, 189, 196, 199
sorting
  data in a grid, 515–520
  direction, 516–520
  expressions, 516–520
  GridView control, 510–512
  using BindGrid method, 513–520
Sorting event handler, 516, 519
sorting query results using ORDER BY, 306–307
Source property, 217
Source View, 151, 442, 476–477
 tag, 31
specific employee details, 344
SQL, 293
  expressions and operators, 310–313
  origins, 293
  reading data from a single table, 294–307
  reading data from multiple tables, 307–310
  stored procedures, 326–330
  transact-SQL functions, 313–318
  updating existing data, 322–326
  working with groups of values, 318–322
SQL injection attacks, 528
SQL queries, 294
  limiting the number of results with TOP, 307
  matching patterns with LIKE, 304–305
  row filtering with WHERE, 302–303
  SELECT statement, 297–299
  selecting certain fields, 299–300
  selecting ranges of values with BETWEEN, 303–304
  selecting unique data with DISTINCT, 300–302
  sorting results using ORDER BY, 306–307
  subqueries, 308–309
  table joins, 309–310
  using the IN operator, 305–306
  viewing results in text format, 299
  wildcard characters in, 304
SQL scripts
  executing, 272
  using, 272
SQL Server 2005 Express Edition, 1, 6, 252
  database administrator for, 22
  downloading, 22
  installing, 22
SQL Server Authentication, 335, 341, 548, 550
SQL Server Management Studio Express, 6, 254, 295
  accessing, 23

changing security settings, 24
changing server settings, 25
database creation, 256–257
downloading, 23
installing, 22–24
managing your database server, 24
number of affected rows affected by a query, 298
starting, 256
to connect to SQL Server, 341
Windows Authentication, 23
SQL Server Setup Wizard, 549
SQL statement, 336
SqlClient namespace
importing, 333–334
SqlCommand class, 333, 427
execution methods, 337
SqlCommand object, 370
adding paramters to, 346
preparing, 336
with parameterized DELETE query, 394
with parameterized INSERT query, 371
with parameterized UPDATE query, 379
SqlCommand object comm
ExecuteCommand method, 339
SqlCommandBuilder object, 521, 523
SqlConnection class, 332
SqlConnection object conn
Open method, 339
SqlDataAdapter, 522
Update method, 521, 523
SqlDataAdapter class, 470, 497, 499
SqlDataAdapter object, 493, 502, 523
properties, 521
SqlDataReader class, 333, 360–361, 427–429, 471
Read method, 343–344
SqlDataReader object, 337, 493
advantages/disadvantages, 469
alternatives to, 470
SqlDataSource object, 470–471, 489
advantages/disadvantages, 492–494
binding to DetailsView, 479–489
binding to GridView, 472–478
generating INSERT, UPDATE and DELETE statements, 480, 489
use with address book, 476
SqlException class, 218
SQRT function, 315
square root function, 315
Src, 131
src attribute, 39
SrteamReader class, 572
StackTrace property, 217
Standard table view, 284
standard web controls, 613–628
standard web server controls, 103–110
Start Page, 150
static method, 578
Static mode (validation), 228
StaticItemTemplate, 197
SteMapDataSource control, 197
stored procedure
basic form, 326
example, 327–330
for DetailsView records, 467
stored procedures, 326–330
use with employee directory, 397–399
StreamReader object, 580–582
StreamWriter class, 572–573
StreamWriter object, 577, 580
string (C#), 60
String (VB), 59–60, 67
string concatination operator, 70
string functions, 315–317
string variables, 581
strings, 59, 61
strongly-typed languages, 61
Structured Query Language (*see* SQL)
style attribute, 137

Style Builder tool, 192
style properties, 138
style rules, 136
  inline, 137
Style Sheet template, 190
style sheets, 135–136
  creating, 190–192
  embedded, 137
  external, 136
styled address book, 440
styles
  classes, 137–138
  CssClass property, 139–141
  editing, 191
  element type selectors, 137
  inline, 137
styling the DataList, 424–426
styling the DetailsView, 450–452
styling the GridView with templates, skins and CSS, 436–440
Sub (VB), 55, 65
subclass
  deriving, 84
Subject (email messages), 594
Subject drop-down list, 365
Submit button, 222–223, 225, 235
Submit Request button, 248, 369, 372, 378
submitButton_Click method, 224, 249, 378, 397
subqueries, 308–309
subroutines
  components, 55
  numbers of parameters, 67
  page events, 56
  web control, 54–56
SUBSTRING function, 316–317
subtraction operator (-), 311
SUM function, 322
survey form
  HTML server control example, 97–101
switch (C#), 71
switch statements, 585
SwitchPage event handler, 124
system databases, 256
System.Configuration namespace, 364
System.Data.SqlClient namespace, 332
System.IO namespace, 571, 576, 586
  groups of classes, 572
System.Net.Mail namespace, 593
System.Web.Security namespace, 532
System.Web.UI.HtmlControls namespace, 96
System.Web.UI.MasterPage class, 133
System.Web.UI.MobileControls namespace, 103
System.Web.UI.Page class, 94
System.Web.UI.UserControl class, 126, 128
System.Web.UI.WebControls class, 86
System.Web.UI.WebControls namespace, 103
System.Windows.Forms namespace, 103

T

table joins, 309–310
table relationships
  and diagrams, 287–292
  creating and visualizing, 287
  many-to-many relationships, 290–292
  one-to-many relationships, 288–290
  one-to-one relationship, 288
Table View, 284
tables
  querying, 298
tables (databases), 253
TagName, 131
TagPrefix, 131
TargetSite property, 217

template styles
designed through CSS, 438
in GridView, 437
TemplateControl class, 85
TemplateField column, 441, 461, 463
templates
access to controls within, 406
DataList control, 405
Repeater control, 355
using in DetailsView, 459
using in GridView, 459–463
terminating a loop, 76
ternary operator, 484
test query
for our data source, 481
Test Query button, 476, 480
text
placement inside a web form, 101
text box, 102
adding to a web form, 101
text editors, 27
text files, 571
reading from, 571, 580–582
writing to, 571, 576–580
Text property, 101, 104, 128
TextBox, 154
for username and password, 536
TextBox control, 104, 128, 578, 602, 627
TextBox controls, 461
widening, 461
TextBox web server control, 101–102
TextMode property, 104
theme folder
cerating, 189
themes, 136, 189, 194
Throw keyword, 215
time, 30
added through ASP.NET code, 27
adding script through C#, 28
adding script through Visual Basic, 28
not displayed in page, 29
timeLabel, 30
timeout attribute, 538
Timeout property, 181
To (address in email message), 594
Toolbox, 154–155, 174
collapsed tabs, 155
controls, 155
Login tab, 544
Standard tab, 154
Validation tab, 229
TOP keyword, 307
ToString, 86
transact-SQL functions, 313
arithmetic functions, 314–315
data and time functions, 317–318
string functions, 315–317
working with groups of values, 318–322
TreeView control, 120–123, 197
events, 643
methods, 643
properties, 640–643
troubleshooting
IIS doesn't load, 14
Try block, 214–216, 354, 377
Try-Catch-Finally blocks, 351, 354, 370
Try-Catch-Finally construct, 206, 215
syntax, 213

U

unauthenticated users (*see* anonymous users)
underscore operator (_), 313
Update button, 386, 459
Update Employee button, 390–397
Update Item button, 416–422
Update link, 463, 485
Update method, 521, 523
UPDATE query, 379, 393, 480
UPDATE statements, 324–325

syntax, 324
updateButton control, 382
UpdateCommand property, 521
UpdateEmployee stored procedure, 421
UpdateItem method, 419–421
updating database records, 378–393
use WHERE statement, 379
updating DetailsView records, 463–468
updating existing data, 322–326
Upload button, 592–593
uploading files, 590–593
uploading files from the client to the server, 572
UPPER function, 315
uppercase, 315
user access
denying/allowing, 539, 559
setting individual rules, 561
user accounts, 534
hard-coded, 535–538
User instance databases, 546
user interaction
with web application, 3
username, 567
editing, 422
entering, 536
storage in authentication ticket, 537
verification, 537
usernameTextBox control, 230
users, 3
creating, 554–556
Users role
assigning, 558
using (C#), 86
Using construct, 577
using statements, 153

V

validating user input, 528
validation controls, 219–250, 528, 628–633
CompareValidator, 231–233
CustomValidator, 239–242
RangeValidator, 233–234
RegularExpressionValidator, 236–239
RequiredFieldValidator, 230
using, 229–242
ValidationSummary, 235–236
validation errors, 236
validation groups, 242–245
default, 245
Validation tab, 229
ValidationExpression, 237
ValidationGroup property, 242
ValidationSummary control, 235–236, 633
Value property, 105
variable declarations, 59
variables, 59
data types, 59–60
initialization, 59
VB, 4, 28, 48
and arrays, 207
arrays, 62
as strongly-typed language, 61
case sensitivity, 71
Click event, 53
code-behind files, 88–89
comments in, 37
data types, 60
declaring an array, 63
Do While loop, 74
editing Default.aspx, 152
enabling Debug Mode, 167
End Sub to mark end of script, 30–31
file upload, 591
For loop, 75
functions, 65
HTML server controls in , 99
If Else statement, 70
operators, 69

page events, 57
reading from a text file, 580
<script> tags in, 28, 37
<script> tags in, 28, 37
selecting case, 71
shared method, 578
simple button and label (code-behind files), 88–89
simple button and label (non code-behind files), 87
standard parentheses for arrays, 64
subroutine components, 55
underscore to mark end of a line, 70
While loop, 72
write-only properties, 129
writing content to a text file, 576
VB.NET code (*see* VB)
View Code, 152
View More details button, 406, 411–413
view state, 40, 44–47
data persistence, 45
disabling, 47, 413
storage of information, 46
View State
for storing data sets, 506–509
ViewDriveInfo method, 584
views, 123
ViewState, 369, 529
ViewState collection, 508
virtual directories, 15–20
changing options, 19
creating, 16–18
default
IISHelp, 15
default document types, 18
Properties, 19
seeing and configuring, 18
subdirectories, 15
testing, 17
virtual directory
as IIS Application, 162
basic settings, 162
Virtual Directory Creation Wizard, 161
Virtual Directory option, 19
virtual directory properties
editing, 162
Virtual Directory tab, 162–163
Virtual Root field, 21
Virtual Web Developer
errors, 210
Visual Basic (*see* VB)
Visual Studio, 143
Visual Web Developer, 143, 146–156
and DataList, 422–424
Code Editor, 151–153
creating a new master page, 196–197
creating a new web site, 147
database connection error, 340
database creation, 255
debug mode, 159–160, 204–210
executing a page, 158
features, 148–156
IntelliSense, 153–154
launching system's default web browser, 159
opening Learning in, 220
permissions, 573
Properties window, 155–156
refreshing display, 166
Solution Explorer, 149–150
Toolbox, 154–155
using with IIS, 165–166
Web Forms Designer, 150–151
web server, 157–160
web server in action, 159
Visual Web Developer 2005 Express Edition, 1, 5
downloading, 26
installing, 25
void (C#), 55, 65

W

Watch window, 207
web applications
  ASP.NET as technology for, 3
  building, 143
  core features, 166–186
  location on computer, 12
  securing, 559–561
  user interacting with, 3
web browsers, 5, 159
  changing, 160
  View Source feature, 31
web client, 3
Web Configuration File, 167
web development projects, 144–146
  executing, 156–166
Web form template, 199
web forms, 94–95
  access and maniupulation, 94
  basic structure, 94
  differences from master pages, 134
Web Forms Designer, 150–151
  code-behind file, 152
  Design View, 150
  Source View, 151
  viewing Toolbox controls, 154
web server, 2–3
  access to, 13
  configuring, 11–21
  installing, 6–9
  starting, 158
  testing in C#, 157
  testing in VB, 157
  Visual Web Developer, 157–160, 163
web server controls, 95, 101–102
  advanced controls, 112–125
  list controls, 110–112
  no direct link to HTML elements they generate, 101
  standard, 103–110
  styling, 192–193
  summary, 102
  tag name preceded by asp:, 101
web site creation, 147
web user controls, 95, 126, 132
  creating, 126–130
  examples, 131–132
  methods, properties, etc., 128
  using, 131–132
  work with code-behind files, 129
Web.config, 150, 163, 166–170
  adding connection string to, 473
  as XML file, 167
  configuration as root element, 169
  customErrors, 212
  database connection strings in, 364
  debugging in, 158
  namespace references, 168
  retrieving configuration data from, 364
  theme attribute, 194
  to store custom information, 168
Web.config file, 534, 551
  authentication mode, 532, 553
    <forms> tag attributes, 538–539
  authorization section, 532, 560
  <authorization> tag, 539–541
  <credentials> tag, 541–542
  storing users in, 541–542
  to store connection strings, 553
  to store general application settings, 553
Web.sitemap files, 120, 197
  limitations, 121
WebControl class
  methods, 612
  properties, 611–612
WHERE clause, 312–313, 321, 325–326, 379, 480, 520
  SQL queries, 302–303, 308, 310–311
WHERE... button, 480

While loop, 342, 580, 582
While loops, 72
  results of, 73
whitespace characters
  trimming, 315
wildcard characters, 304
Windows Authentication, 335
Windows authentication, 531
Wizard control, 125
Write button, 578
WriteOnly modifier, 129–130
write-only properties, 129–130
WriteText method, 576
writing to text files, 571, 576–580
  permissions, 573–575
wwwroot folder, 13
  web server access to files in, 13
WYSIWYG interface, 150

X

XML basics, 117
Xml control, 628
XmlDataSource object, 470

Y

YEAR function, 318

Z

zero-based arrays, 64
zooming, 282