About the Author

Ken Cox is a Canadian writer and programmer. He earned a Bachelor of Applied Arts (BAA) degree in Radio and Television Arts from Ryerson University in Toronto, which led to a 25-year career as an on-air journalist in Toronto and Quebec City. He contributed reports to local stations CFRB and CJAD as well news networks such as CBC, CBS, NBC, and the BBC. His claim to fame is that he has chatted in English and French with Queen Elizabeth II of England.

Ken’s passion for computers and things high-tech led him to earn a college certificate in Technical Communications. He then pursued a second career as a technical writer and information developer with companies including Nortel in Toronto. His documentation has won numerous awards from the Society for Technical Communication.

As the Internet and World Wide Web became popular, Ken started tinkering with Web pages. Despite having no formal training in programming, he found himself part of the earliest beta of a ground-breaking Microsoft product that was code-named Denali. Denali became Active Server Pages (ASP) 1.0, which later evolved into ASP.NET with Visual Studio and Visual Web Developer as its primary development tool.

Microsoft has awarded Ken its coveted Most Valuable Professional (MVP) status each year since 1998 in recognition of his volunteer assistance to users in online communities such as the ASP.NET newsgroups.

He currently works as a contract Web applications consultant, programming writer, technical reviewer, author, and as a contributing editor for Visual Studio Magazine.

Ken, his wife Vilia, and their dog Goldie (a GoldenDoodle) spend spring, summer, and fall at a peaceful lakefront home in a forest in Nipissing Township, Ontario, Canada. They winter in Victoria, British Columbia.
Dedication

To my wife, Vilia, for encouraging me to pursue my dreams.
Author’s Acknowledgments

Thanks to Acquisitions Editor Katie Feltman for showing faith in my abilities by offering me a chance to write a book of my own on a topic I love. To Rebecca Senninger and Blair Pottenger, the project editors: thanks for being my air traffic controllers, troubleshooters, advisors, and all-round publishing resources. Thanks also to my technical reviewer and fellow MVP, Mark Rae, for catching my slips and making valuable suggestions for a better book. The mistakes that remain are mine.

To my wife, Vilia: You’ve been a tremendous support for over 37 years. This book is just another example of how I couldn’t manage without your love and guidance. You’ve always encouraged me to follow my dreams. I’m a lucky guy to have found you.

Finally, a shake of the paw and a “bikkie” (dog biscuit) for our dog Goldie. Your muzzle nudges and refusal to be ignored guarantee restorative breaks outdoors at 12:30 p.m., 5:30 p.m., and 10:30 p.m. every day, rain or shine. Okay, Goldie, go find your ball!
Publisher's Acknowledgments

We're proud of this book; please send us your comments through our online registration form located at www.dummies.com/register/.

Some of the people who helped bring this book to market include the following:

Acquisitions, Editorial, and Media Development
- **Project Editor:** Rebecca Senninger
- **Senior Acquisitions Editor:** Katie Feltman
- **Copy Editor:** Brian Walls
- **Technical Editor:** Mark Rae
- **Editorial Manager:** Leah Cameron
- **Editorial Assistant:** Amanda Foxworth
- **Sr. Editorial Assistant:** Cherie Case
- **Cartoons:** Rich Tennant (www.the5thwave.com)

Composition Services
- **Project Coordinator:** Erin Smith
- **Layout and Graphics:** Stacie Brooks, Reuben W. Davis, Alissa D. Ellet, Shawn Frazier, Christine Williams
- **Proofreaders:** Cynthia Fields, John Greenough, Bonnie Mikkelson
- **Indexer:** Infodex Indexing Services, Inc.

Special Help
- Teresa Artman; Kelly Ewing; Virginia Sanders

Publishing and Editorial for Technology Dummies
- **Richard Swadley**, Vice President and Executive Group Publisher
- **Andy Cummings**, Vice President and Publisher
- **Mary Bednarek**, Executive Acquisitions Director
- **Mary C. Corder**, Editorial Director

Publishing for Consumer Dummies
- **Diane Graves Steele**, Vice President and Publisher
- **Joyce Pepple**, Acquisitions Director

Composition Services
- **Gerry Fahey**, Vice President of Production Services
- **Debbie Stailey**, Director of Composition Services
Contents at a Glance

Introduction ................................................................. 1

Part I: Getting to Know ASP.NET and Visual Web Developer ................................................... 7
Chapter 1: Understanding Microsoft’s Web Technologies ................................................................. 9
Chapter 2: Getting Up and Running ..................................................................................................... 21
Chapter 3: Creating a Useful ASP.NET Site ...................................................................................... 37
Chapter 4: Managing Data and Other CRUD .................................................................................. 47
Chapter 5: Handling User Input and Events ...................................................................................... 61

Part II: Immersing Yourself in Data...................................................................................... 75
Chapter 6: Fetching and Presenting Data with SqlDataSource .......................................................... 77
Chapter 7: LINQ as a Data Language ................................................................................................. 99
Chapter 8: Using LINQ to SQL and the LinqDataSource ................................................................ 121
Chapter 9: Creating and Consuming Diverse Data ......................................................................... 143

Part III: Enhancing the Interface and User Experience ............................................................. 163
Chapter 10: Common Elements: Style Sheets, Master Pages, and Skins ........................................... 165
Chapter 11: Adding Navigation with TreeView, Menu, Breadcrumb, and SiteMap ................................ 179
Chapter 12: Web Standards, Page Layout, and Usability .................................................................. 193
Chapter 13: Designing the ListView and Other Templated Controls ............................................. 207
Chapter 14: Dynamic Effects, Images, and Rollovers ................................................................. 223
Chapter 15: Enhancing Pages with the AJAX Control Toolkit ....................................................... 239
Chapter 16: Creating and Displaying Rich Content ........................................................................... 255

Part IV: Tracking Users, Controlling Access, and Implementing Security ................................ 271
Chapter 17: Site Security Using Authentication and Membership .................................................. 273
Chapter 18: Creating a Shopping Cart with Profiles ........................................................................ 291
Chapter 19: Validation in Depth ......................................................................................................... 315
Part V: Getting the Bugs Out and Handling Runtime Errors...........................................331
Chapter 20: Debugging and Tracing Pages.................................................................333
Chapter 21: Avoiding Crashes by Handling Exceptions ............................................349
Chapter 22: Ten Tips on Deploying Your Web Application.......................................365
Chapter 23: Ten Tips to Success with ASP.NET .........................................................379
Index ..........................................................................................................................385
Table of Contents

Introduction...................................................................................................................................... 1

I Know Who I Am: Who Are You?.................................................................................................. 1
Less Code, More Productivity ......................................................................................................... 2
How to Use This Book .................................................................................................................. 3
How This Book Is Organized........................................................................................................ 3
Part I: Getting to Know ASP.NET and Visual Web Developer .................................................. 3
  Part II: Immersing Yourself in Data ............................................................................................. 3
  Part III: Enhancing the Interface and User Experience ............................................................ 4
  Part IV: Tracking Users, Controlling Access, and Implementing Security .............................. 4
  Part V: Getting the Bugs Out and Handling Runtime Errors .................................................... 4
Part VI: The Part of Tens .............................................................................................................. 4
What’s on the Web Sites .................................................................................................................. 5
Icons Used in This Book .............................................................................................................. 5
Where to Go from Here .................................................................................................................. 6

Part I: Getting to Know ASP.NET and Visual Web Developer .................................................. 7

Chapter 1: Understanding Microsoft’s Web Technologies .......................................................... 9
  Introducing the Content-Creation Tools ..................................................................................... 9
    Microsoft Office (Including Word 2007)... ................................................................................ 10
    Expression Web ....................................................................................................................... 10
    Expression Blend .................................................................................................................... 10
    Visual Web Developer (Including Express) ........................................................................... 11
  Meeting the Technologies behind Web Applications ............................................................... 12
    Microsoft’s .NET 3.5 Framework ............................................................................................ 12
    ASP.NET 3.5 ........................................................................................................................... 12
    ASP.NET Futures ..................................................................................................................... 13
    ASP.NET 3.5 Extensions ........................................................................................................ 13
    Web services ........................................................................................................................... 14
    JavaScript and client-side code .............................................................................................. 14
    ASP.NET AJAX ....................................................................................................................... 15
    Dynamic HTML ....................................................................................................................... 16
    Extensible Markup Language (XML) ..................................................................................... 17
    Silverlight ............................................................................................................................... 17
    Language Integrated Queries (LINQ) .................................................................................... 18
    ADO.NET ............................................................................................................................... 19
    SQL Server ............................................................................................................................. 19
    Internet Information Services ................................................................................................ 19
Chapter 2: Getting Up and Running ..................................................21

Installing Visual Web Developer Express ..................................................21
Finally! Creating an ASP.NET Web Page ..................................................26
Starting the IDE ..................................................................................26
Creating an ASP.NET Web site .............................................................26
Adding an ASP.NET control ................................................................28
Previewing a page in the browser ........................................................28
Tweaking Your Development Environment ..................................................29
Showing all settings ............................................................................29
Unhiding advanced members .............................................................29
Starting pages in Design view .............................................................30
Working with the Toolbox ......................................................................30
Auto Hide and the pushpin ..................................................................30
Adding controls to the VWDE Toolbox ...............................................31
Peering into a Wall of Windows ...........................................................32
Organizing files with Solution Explorer ...............................................32
Setting Properties in the Properties window ........................................33
Viewing what the Properties window has generated ..............................35

Chapter 3: Creating a Useful ASP.NET Site ..............................................37

Creating the DVD Web Project .............................................................37
Using a SQL Server Express Database ......................................................38
Adding a database to the project ........................................................38
Adding a table to the database ...........................................................39
Generating a Data-Driven Web Page ........................................................43
Adding a single file model Web page ..................................................43
Using the database to build a Web page ...............................................44
Previewing and reviewing the database-generated page ...........................45

Chapter 4: Managing Data and Other CRUD ............................................47

Working with Smart Tags and Designers ................................................48
Showing the Smart Tag and tasks via a menu .........................................48
Using the Smart Tag button ..................................................................48
Enhancing the GridView Control ..........................................................49
Adding a dash of color to the GridView control .....................................49
Sorting, editing, and deleting with the GridView ....................................50
Formatting the date display ................................................................52
Introducing the FormView Control .........................................................53
Adding a FormView control to the page ...............................................53
Changing the FormView control’s templates .........................................54
Using the FormView control to insert a row .........................................56
Analyzing problems with the date input ...............................................58
Validating the date input .....................................................................58
Fixing the Page Title ............................................................................59
Improving Performance with the AJAX Update Panel ..............................60
## Chapter 5: Handling User Input and Events ................................. 61

- Accepting Data in a TextBox Control ........................................ 61
- Creating a regular text box ..................................................... 62
- Accepting passwords (somewhat) securely ............................. 62
- Capturing text with MultiLine mode ........................................ 62
- Allowing creativity with rich text ............................................ 63

## Pushing for Choices with the RadioButton Control .................. 63

- Collecting RadioButtonList Controls ...................................... 64
  - Creating the basic page interface ......................................... 64
  - Adding list items with a Collection editor ............................ 65
  - Capturing the survey choice ............................................. 66

## Checking CheckBox and CheckBoxList Controls ....................... 67

- For Each and the collection .................................................. 69

## Using the DropDownList Control ............................................. 69

- Understanding namespaces ................................................... 71
- Retrieving a list of colors ..................................................... 71
- Displaying the color name and showing the color .................. 71

## Getting Multiple Choices from a ListBox ............................... 72

- Understanding ASP.NET Forms ............................................. 73

## Part II: Immersing Yourself in Data ......................................... 75

## Chapter 6: Fetching and Presenting Data with SqlDataSource ...... 77

- Connecting to SQL Server Express ........................................ 77
- Checking whether SQLExpress is running .............................. 77
- Finding a copy of the Northwind database ............................. 78
- Adding the Northwind database to your application .............. 78
- Connecting to the database .................................................. 79

## Using the SqlDataSource Control .......................................... 81

- Adding and configuring a SqlDataSource control .................... 81
- Consuming Data with the DetailsView Control ....................... 84

## Using Parameters in Queries ................................................. 86

- Getting a parameter value from a TextBox control ................ 86
- Returning the country names with no repeats ....................... 88
- Filling a drop-down list with data from a SqlDataSource .......... 89
- Changing the parameter source .......................................... 89
- Obtaining a parameter from a Session variable ...................... 90
- Passing a parameter on a query string .................................. 92

## Creating a Master/Detail Page ............................................... 93

- Designing the page layout ................................................... 94
- Fetching data for the master ............................................... 95
- Fetching data for the details .............................................. 96
- Configuring the GridView and DetailsView controls .............. 97
Chapter 7: LINQ as a Data Language ..............................................99
  Setting Up the LINQ Examples.................................................................99
  Creating the DataContext object .................................................................100
  Creating ASP.NET pages for the examples ...................................................100
  LINQing with From, Where, and Select .....................................................101
    Targeting the source in a From...In clause ............................................102
    Narrowing the thingies with a Select clause .........................................102
    Filtering with a Where clause ..............................................................103
  Filtering with an Eye on Strings ..............................................................104
    Choosing what you Like ........................................................................104
    Investigating what the query Contains() ..............................................105
    It all StartsWith() and EndsWith() strings ..........................................105
  Filtering Based on Numbers .................................................................106
    Finding expensive items ........................................................................106
    Filtering dates and times .......................................................................107
  Thoroughly Aggregating Data .................................................................107
    Just give me the list and the Count() ......................................................107
    If at first you don’t succeed, you’re running about Average() ..............108
    First the Dim and then the Sum() ...........................................................109
    Returning the Min() and the Max() values ..........................................109
    Stepping along with Skip() and Take() ..................................................110
  Grouping, Sorting, and Making Distinct ..................................................111
    Creating the language grouping page ......................................................111
    Analyzing the LINQ grouping query .......................................................113
    Rendering grouped data on a Web page .................................................114
  Using LINQ to Create and Query XML ...................................................115
    Creating the KinFolk class .......................................................................115
    Using object initializers to add data .......................................................116
    Building the XML file with LINQ to XML ..............................................117
    Filtering XML with a LINQ to XML query ...........................................119

Chapter 8: Using LINQ to SQL and the LinqDataSource .....................121
  Building a LINQ to SQL CRUD Page .......................................................121
    Creating the database access code .........................................................122
    Hooking up with the LinqDataSource control .........................................123
    Creating the user interface with a ListView .........................................123
    Using LINQ to work around a deletion constraint ..................................124
    Confirming deletion requests ..................................................................126
  Enhancing Usability with LinqDataSource ...............................................127
    Putting a name to a number ....................................................................127
    Allowing users to select from a drop-down list .......................................128
  Filtering Data with LinqDataSource .........................................................131
    Creating a LinqDataSource to fetch categories .....................................131
    Adding a drop-down list and connecting it to the LinqDataSource ..............131
    Filtering the LinqDataSource with a Where parameter .........................132
Displaying Hierarchical Data with LINQ....................................................133  
Grouping with a LINQ query........................................................................133  
Creating the outer GridView control..........................................................135  
Adding a Label control to display categories ..........................................136  
Creating the inner GridView control..........................................................137  
Updating Data with a LINQ Query...............................................................137  
Exclaiming with an Extension method......................................................138  
Building a page to update product data.....................................................139  
Inserting Data with the DataContext ..........................................................140

Chapter 9: Creating and Consuming Diverse Data  .......................143  
Putting an RSS Feed on a Page.................................................................143  
Analyzing an RSS feed................................................................................143  
Using the XmlDataSource control..............................................................144  
Displaying XML data by using the DataList ..............................................145  
Making an RSS Feed Available from Your Site .......................................146  
Transforming XML Data into HTML Markup............................................148  
Gathering the source XML data.................................................................149  
Creating the XSL style sheet........................................................................149  
Using the ASP.NET Xml control.................................................................150  
Connecting Web Applications to an Access Database..............................151  
Creating a Simple Web Service.................................................................152  
Adding a Web Reference to a Project..........................................................155  
Creating a Page to Use the Web Service....................................................156  
Creating a Daylight Saving WCF Service....................................................157  
Creating the Service Consumer Web Form...............................................159  
Connecting to a WCF Endpoint..................................................................160

Part III: Enhancing the Interface and User Experience......................163

Chapter 10: Common Elements: Style Sheets  
Master Pages, and Skins .................................................................165  
Deciding Where Style Rules Belong..........................................................165  
Quick and not-too-dirty with AutoFormat...............................................166  
Keeping styles close and inline................................................................166  
Storing styles in the page’s <style> tag....................................................167  
Storing styles in an external CSS style sheet...........................................167  
Using the VWD Style Sheet Tools...............................................................168  
Attaching an external style sheet...............................................................168  
Adding a style rule to an external style sheet............................................169  
Splashing on some wild style....................................................................170  
Applying a style to a TextBox control......................................................171  
Analyzing the generated style.................................................................172
Managing Style Rules ........................................................................................................... 172
Moving styles from a page to a style sheet ................................................................. 173
Adding, modifying, and deleting styles ........................................................................ 174
Using Master Pages with Slavish Devotion .............................................................. 174
Creating a master page ................................................................................................. 174
Adopting a master page while creating a regular page .............................................. 175
Skinning Is Just What It Themes .................................................................................. 176
Creating a theme for GoGreen .................................................................................... 176
Assigning a theme to the whole Web site ................................................................. 177
Assigning a theme to an individual page ..................................................................... 178

Chapter 11: Adding Navigation with TreeView, Menu, Breadcrumb, and SiteMap .......................................................... 179

Using a Treeview on a Web Page .................................................................................. 179
Creating TreeView nodes in the designer .................................................................. 180
Creating a Web.sitemap file for navigation data ....................................................... 182
Generating a treeview from a Web.sitemap file ......................................................... 183
Using the treeview with an XMLDataSource control ............................................... 184

Building a Menu for Your Site .................................................................................... 186
Creating a menu in the designer .................................................................................. 187
Generating a menu from a Web.sitemap file .............................................................. 188

Adding a Breadcrumb Feature to Your Pages ............................................................. 190
Creating a breadcrumb on a master page .................................................................... 190
Customizing a breadcrumb .......................................................................................... 191

Chapter 12: Web Standards, Page Layout, and Usability ........................................ 193

Choosing an HTML Flavor ............................................................................................ 193
Visual Web Developer and standards .......................................................................... 194
External XHTML validation .......................................................................................... 196

Creating Columns Using CSS Float ............................................................................ 196
Divvy up the page with <div> tags .............................................................................. 198
Document Outline lays out the structure .................................................................... 199
Dedicated style rules and float: left ............................................................................. 199

Reducing Load Times and Improving Performance .................................................. 200
Turning off ViewState .................................................................................................. 200
Caching “expensive” content ...................................................................................... 201

Meeting Accessibility Requirements ........................................................................ 202
Alternate text for images ............................................................................................. 203
Avoiding output as tables ........................................................................................... 203
Is client script allowed? ............................................................................................... 204
Validating Web accessibility ......................................................................................... 204

Increasing a Page’s Usability ....................................................................................... 204
Setting the tab order .................................................................................................... 205
Adding access/accelerator/shortcut keys ................................................................... 205
Setting the focus on startup and default buttons ..................................................... 206
Chapter 13: Designing the ListView and Other Templated Controls . . . 207

Understanding Templated Controls ...........................................................207
Repeating yourself with the Repeater ..................................................208
Letting the designers generate templates ........................................210
Rolling Your Own with the ListView Control .........................................212
Generating the DataContext ..................................................................212
Configuring the LinqDataSource ..........................................................213
Setting up the ListView ......................................................................214
Adding the mandatory LayoutTemplate .............................................214
Displaying data with ItemTemplate ..................................................215
Editing records with EditItemTemplate ............................................216
Adding records with InsertItemTemplate ......................................218
Advising users there’s no data with EmptyDataTemplate ..................219
Using the ItemSeparatorTemplate .....................................................220
Making a horizontal list with flow ......................................................220
Using the DataPager with a ListView ..................................................221

Chapter 14: Dynamic Effects, Images, and Rollovers ................. 223

Creating Rollover Effects .....................................................................223
Making a text rollover with a stylesheet ...........................................223
Using JavaScript and images for rollovers ......................................225
Creating and Displaying Graphics on the Fly ........................................227
Generating a custom image in ASP.NET ..............................................228
Updating and displaying the custom image ......................................231
Displaying Uploaded Image Files As Thumbnails ..................................232
Accepting a file upload ......................................................................232
Creating a thumbnail image WebHandler ............................................236
Displaying an uploaded image as a thumbnail ....................................238

Chapter 15: Enhancing Pages with the AJAX Control Toolkit ........ 239

Introducing the AJAX Control Toolkit ................................................239
Automatically Completing Data As the User Types ...............................241
Preparing the word list ......................................................................241
Creating the data lookup Web service ..............................................242
Creating the data lookup page ..........................................................243
Helping Users Understand What to Enter ...........................................244
Enhancing a text box with the TextBoxWatermarkExtender ............245
Adding style to a watermark .............................................................245
Guiding Input with a Masked Text Box .............................................246
Creating a masked input ...................................................................246
Using masks and custom characters ...............................................247
Choosing Dates with a Calendar .........................................................249
Positioning Content to Stay on Top .....................................................251
Creating a floating style ....................................................................251
Adding Panel controls to make <div>s .............................................252
Adding the AlwaysVisibleControlExtender on a page ....................253
Chapter 16: Creating and Displaying Rich Content ........................................... 255
  Creating Your First Rays of Silverlight.......................................................... 255
  Setting up the Web project ............................................................................... 256
  Creating static XAML content......................................................................... 258
  Embedding Silverlight with the ASP.NET Silverlight Control ....................... 259
    Hosting Silverlight with the ASP.NET Silverlight control.......................... 260
    Playing Windows Media files in Silverlight............................................... 262
  Displaying Rich Media with the MediaPlayer Control.................................... 263
  Embedding Flash in an ASP.NET Page ............................................................. 264
    Downloading and installing Flasher.............................................................. 264
    Using the Flasher control on a page............................................................. 265
  Ensuring Accurate Rendering with PDF........................................................... 266
    Rendering PDF within the browser page...................................................... 266
    Rendering PDF within a new browser page.................................................. 266
    Forcing the Open or Save dialog box............................................................ 267
  Serving Word on the Web.................................................................................. 268

Part IV: Tracking Users, Controlling Access, and Implementing Security .......... 271

Chapter 17: Site Security Using Authentication and Membership ........................... 273
  Understanding Authentication.......................................................................... 273
  Preparing a Site for Membership..................................................................... 274
    Obtaining the Small Business Starter Kit..................................................... 274
    Installing the Small Business Starter Kit..................................................... 274
    Determining the requirements...................................................................... 275
  Creating the Membership Database.................................................................. 275
    Configuring forms authentication................................................................. 276
    Creating and enabling a role......................................................................... 277
  Implementing Registration and Login .............................................................. 278
    Creating the Registration page with CreateUserWizard............................ 278
    Creating the Login page................................................................................ 280
    Creating the Password Recovery page......................................................... 281
    Configuring the SMTP (Mail) settings......................................................... 282
    Creating a Change Password page............................................................... 283
    Providing a Login/Logout link...................................................................... 284
  Adding an Administration Area........................................................................ 284
    Adding the Admin folder and a page............................................................ 285
    Building the Membership List page............................................................. 285
  Applying Roles and Security............................................................................ 286
    Securing the Admin folder with roles.......................................................... 286
    Understanding access rules........................................................................... 287
    Adding an administrator............................................................................... 288
Chapter 18: Creating a Shopping Cart with Profiles .............................................. 291
Introducing ASP.NET Profiles ........................................................................... 291
Setting Up the Small Business Sample Site ..................................................... 292
Previewing the Final Web Interface ............................................................... 293
  The Add to Cart interface ............................................................................... 293
  Tracking the cart status ................................................................................. 293
  Gawking at the cart contents ......................................................................... 294
Building the Shopping Cart in Code ............................................................... 294
  Defining a shopping cart item class .............................................................. 294
  Defining the shopping cart class ................................................................... 299
  Enabling profile data and anonymity in web.config .................................... 305
Updating a Web Page to Add Profile Data ...................................................... 306
  Inserting a LinkButton into the page ........................................................... 306
  Configuring the LinkButton control ............................................................. 307
  Adding the LinkButton event handler ......................................................... 308
Building a Page to Manage Cart Contents .................................................... 309
  Adding the shopcart.aspx page .................................................................. 309
  Adding an ObjectDataSource to handle data .............................................. 309
  Adding a GridView and using the ObjectDataSource .................................. 311
  Creating a Calculations class ...................................................................... 311
  Inserting Calculations columns .................................................................. 313
Walking Through the Shopping Cart Profile .................................................. 314
  Adding items to the cart .............................................................................. 314
  Updating the quantity of an item .................................................................. 314

Chapter 19: Validation in Depth ....................................................................... 315
Remembering User Input Is Evil ...................................................................... 316
Forcing the User to Enter Something .............................................................. 316
Ensuring That a Value Is within a Range ......................................................... 317
Checking and Comparing Values .................................................................... 319
  Comparing values in two controls ............................................................... 319
  Making the CompareValidator dynamic ..................................................... 320
  Checking a data type .................................................................................. 321
Using the RegularExpressionValidator ......................................................... 322
  Testing for one, two, or three numbers ....................................................... 322
  Checking the length of text in a multiline text box ..................................... 323
Validating Data with Code ............................................................................... 324
Validating by Groups ...................................................................................... 326
Displaying a Summary of Invalid Fields ......................................................... 327
Defanging Markup for Safety ......................................................................... 328
Part V: Getting the Bugs Out and Handling Runtime Errors

Chapter 20: Debugging and Tracing Pages

Setting Up an Error Page Scenario
Analyzing Design-Time Errors
Discovering Compile-Time Errors
  Building a single page
  Building a whole Web site with exclusions
Finding Logic Errors
  Analyzing the sample page at runtime
  Setting a breakpoint in the code
  Examining values while debugging
Tracking Down a Runtime Error
Breaking Based on a Condition
Editing a Value during Execution
Panes to Ease the Pain
Tracing the (Mis)Steps of a Web Page
  Implementing trace in a page
  Implementing trace for a whole site
Using the Debugger Keys and Toolbar

Chapter 21: Avoiding Crashes by Handling Exceptions

Understanding Exceptions and Their Messages
Global Error Handling
Catching and E-Mailing Exceptions
Using Try...Catch in Risky Situations
Executing a Statement, Finally
Some Common Error Messages and Where to Look
  System.NullReferenceException
  Are you missing an assembly reference?
  ‘Button1_Click’ is not a member of ‘ASP.default2_aspx’
  Expression of type ‘1-dimensional array’ is not queryable

Chapter 22: Ten Tips on Deploying Your Web Application

Use the Copy Web Site Tool
Connecting via FTP
Connecting by using the FrontPage extensions
Connecting via the file system
Transferring files in the Copy Web tool
Use the SQL Publishing Wizard
Creating a database script
Creating a remote database from a script
Copy a SQL Express Database
Fix the @#$%^& SQL Connection
Choose an ASP.NET-Friendly Host ............................................................. 374
Head Off a Serious Lack of Trust .............................................................. 374
Arrggh! It Works Fine on MY Machine! .................................................... 374
Gather Troubleshooting Info ................................................................. 375
Precompile If You’re Code Shy ............................................................... 376
Encrypt Connection Information ........................................................... 377

Chapter 23: Ten Tips to Success with ASP.NET  . . . . . . . . . . . . . . . . . . 379
Stop Bashing Your Head against a Wall ................................................... 379
Google Is Your Friend ........................................................................... 380
Read the Reference Documentation ..................................................... 380
  Built-in online help ........................................................................ 380
  Web-based reference material ......................................................... 380
Ask a Good Question, Get a Good Answer ............................................. 381
Get Free Peer-to-Peer Support .............................................................. 381
  Join forums.asp.net .................................................................... 381
  Find experts at msnews.microsoft.com ........................................ 382
Use the Starter Kits .............................................................................. 382
Read the Hottest Blogs ....................................................................... 382
Watch the Videos ................................................................................. 383
Visit the Expert Web Sites ................................................................. 383
Use the Free Tools .............................................................................. 384

Index .................................................................................................. 385
Greetings! You just entered the world of *ASP.NET 3.5 For Dummies*. In case you weren’t told on the way in, ASP.NET is Microsoft’s technology for building dynamic, interactive, data-driven Web pages. The primary tool for creating ASP.NET sites is Visual Web Developer (VWD), which you use throughout this book.

Wait a minute! An introduction to an introduction is not only wordy and redundant, it’s superfluous and unnecessary.

**I Know Who I Am: Who Are You?**

My full name is Kenneth John Cox. I was born in Windsor, Ontario, Canada. I’m a former broadcast journalist (the pejorative term is *spit-collector*) whose hobby (long ago) was playing with computers. Somehow, I learned enough about ASP.NET to get paid for creating Web applications. When they pay you for your hobby, it sure beats working for a living!

Here’s what I assume about you, gentle reader:

- You use a computer and know your way around Windows XP or Windows Vista.
- You’re familiar with the World Wide Web and can connect to the Internet.
- You’ve created a Web page in a tool like FrontPage or Dreamweaver and probably know some HTML markup.
- You grasp basic programming concepts. The terms *variable* and *loop* don’t frighten you — but you aren’t necessarily a programmer.

You may have any number of reasons for digging into this ASP.NET book:

- You volunteered to create a statistics Web site for your kid’s soccer league.
- You’re putting your home-based business on the Web and need a data-driven page.
You develop Web sites on platforms like Java and PHP and want to make yourself more marketable by including Microsoft’s technology.

You’ve worked with a previous release of ASP.NET and want to get up to speed on new stuff like AJAX, LINQ, and the ListView control.

Your boss is dabbling in ASP.NET and might let you play in his sandbox if you talk a good enough game.

You collect For Dummies books and master each book’s subject before moving to the next one.

**Less Code, More Productivity**

When I agreed to write a book from scratch on ASP.NET 3.5, I made it clear that I wanted it to be very hands-on and task-oriented. I show you how to use Microsoft’s latest graphical tools — designers, editors, and wizards — to their best advantage. Instead of treating new features like AJAX and LINQ as separate add-ons, I integrate them into many samples.

Some professional developers would have you believe that the only effective way to create ASP.NET pages is to write the code by hand. (Do the words real men and quiche ring a bell here?) Their geeky noses have been stuck to the keyboard for so long they’ve been left behind. Microsoft has implemented powerful design-time tools in Visual Web Developer, so why not use them to be more productive?

Wherever possible, I favor the drag, drop, choose, and configure methods over typing code. Here’s why:

- **It’s faster.** You don’t have to know — or even understand — the ins and outs of every object before creating something useful.

- **You create fewer bugs.** Microsoft’s built-in designers write quality code based on your choices.

- **Pages are easier to maintain.** Programmers are notorious for failing to document what their code performs and many insist that code is “self-documenting.” When you revise someone else’s code by rerunning a wizard, you spend less time playing catch-up.

That said, in many instances in this book, you do write code. Each time, I explain what the code is performing. Don’t fear being overwhelmed if you’re not a code jockey. Everyone’s a beginner at some point.

The book’s code examples are in Visual Basic .NET because Visual Basic is easy to understand, not case-sensitive, and just as powerful as C# when compiled. (Not to mention that I like VB best!)
How to Use This Book

People have different learning styles. Many are adventurers who turn to manuals only to get out of trouble. They barge into a new programming task like a deer into the forest until some grimy detail stops them in their tracks. Suddenly, progress can’t be made until they find an example or fill a knowledge gap. That’s when they scout out a likely topic in the book’s index, follow a few numbered steps, and snatch a snippet of “just-in-time” information.

In contrast to the adventurers, you might be the organized and methodical type. Perhaps you prefer to get a feel for the subject, ease into it, and analyze examples while you’re building skill and confidence. This book accommodates both approaches by including multiple hooks and starting points.

How This Book Is Organized

This book organizes the topics in parts with each part covering a different aspect of creating ASP.NET applications.

Part I: Getting to Know ASP.NET and Visual Web Developer

Part I introduces the technology and contains the information you need to start creating your first ASP.NET pages. The goal is to become comfortable enough with the terminology and tools so you relax in the rest of the book. If you’ve worked with a previous version of ASP.NET and Visual Web Developer, you might want to skim or skip Chapter 1. Chapter 2 is necessary only if you’ve never worked in a Visual Studio or Visual Web Developer environment. In Chapters 3, 4, and 5, I introduce key concepts and ensure your initial success in creating pages that work with user input.

Part II: Immersing Yourself in Data

In Part II, I walk you through the integration of data with ASP.NET pages. Chapter 6 covers the basic needs of virtually every data-driven site using the SqlDataSource control. Don’t miss Chapters 7 and 8, where I cover the new Language Integrated Query (LINQ) features. Chapter 9 digs into other data sources, such as XML and Web services.
Part III: Enhancing the Interface and User Experience

In Part III, you explore the presentation aspects of Web pages. In Chapter 10, I show you how to use the tools and techniques in VWD to create user interfaces. Every site with more than one page needs navigation, and that’s covered in Chapter 11. Chapter 12 looks at HTML standards and how to use a style sheet to divide a Web page into columns. For sophisticated formatting, Chapter 13 walks you through the versatile new ListView control. In the remaining chapters in Part III, you add dynamic effects, boost page response with AJAX, and introduce rich content, such as Microsoft Silverlight, into your pages.

Part IV: Tracking Users, Controlling Access, and Implementing Security

Part IV is largely about security and recognizing returning visitors. In Chapter 17, I show you how easy it is to secure pages by using ASP.NET’s built-in authentication and membership features. The chapter offers professional touches that users appreciate. In Chapter 18, you build an e-commerce style shopping cart by using ASP.NET’s built-in Profiles feature. Chapter 19 demonstrates ways to ensure that users — friendly or otherwise — provide your application with clean, safe, validated data.

Part V: Getting the Bugs Out and Handling Runtime Errors

Turn to the chapters in this part to figure out why a page or site isn’t behaving the way it should. Chapter 20 shows techniques for checking what’s going on deep in your app. Chapter 21 provides defenses to cope with unforeseen errors in a deployed page.

Part VI: The Part of Tens

In Chapter 22, you copy your ASP.NET pages and associated files to the Internet. The last chapter of the book points you toward helpful resources for when you’re stuck or you need to expand your expertise and investigate more complex subjects.
What’s on the Web Sites

This book has two Web sites to provide online resources. The first is the book's official page at www.dummies.com/aspdotnet35fordummies where you can read excerpts, download the book's source code, and fill a shopping cart with extra copies of ASP.NET 3.5 For Dummies for your friends, loved ones, and coworkers.

The second site, www.kencox.ca, is the place for book-related help. It’s my personal site (could you guess by the domain name?) with updated links to tools, forums, and resources that I discuss in this book. There’s a frequently asked questions area, a contact form, errata (hardly any!), and pictures of my dog. Don’t miss the junk drawer-like Stuff section!

Icons Used in This Book

You find a handful of useful icons in this book. Here’s what they mean:

**Tips** highlight a handy shortcut or help you understand something important about ASP.NET or Visual Web Developer.

**This icon** marks something that might trip you up the next time you encounter it.

The Technical Stuff icon alerts you to information (such as a discussion about code) that’s heavier than usual. Skip it if you want and come back when you’re ready.

Prepare to roll your eyes, smirk, or shake your head in disbelief at something that doesn’t make sense.

The Warning icon is like a yellow caution sign on the highway. By not heeding this advice, you could lose data or lead someone to think you don’t know what you’re doing.
Where to Go from Here

If you’re still reading this introduction, you’re the calm, persistent type who wants the A-to-Z story — proceed to Chapter 1. If you landed here while flitting about the book, you’re an adventurer who should try Chapter 4. Interested in LINQ? Jump now to Chapter 7 and play with some queries!
Part I

Getting to Know

ASP.NET and

Visual Web

Developer

The 5th Wave

By Rich Tennant

“What I’m looking for are dynamic Web applications and content, not Web innuendoes and intent.”
In this part...

In this part, especially in Chapter 1, you dive into the technologies that create Web sites on Microsoft’s platform. I include steps for software installation (Chapter 2) and for site creation (Chapter 3), which ensure you’re not flopping around like a fish out of water while you get your feet wet. Help! I’m drowning in metaphors!

In Chapter 4, you create data-driven ASP.NET pages — something you do often as a .NET Web developer. Building on your success, the last chapter (Chapter 5) walks you through assembling forms that accept user input with ASP.NET server controls.

It’s not unusual to feel your head swimming while you wade into a new technology. If something seems over your head, keep dog-paddling as best you can. Remember: The life-guard also started in the shallow end of the pool — and she ended up high and dry! (Okay, I’m done.)
Chapter 1

Understanding Microsoft’s Web Technologies

In This Chapter
- Exploring Microsoft’s tools for creating Web pages
- Understanding the technologies behind dynamic content
- Delving client-side and server-side programming
- Pinpointing the roles of LINQ, DHTML, XML, XAML, and AJAX
- Deciphering postbacks and page refreshes

In the beginning, the World Wide Web (WWW) was flat. It was an electronic library where academics and scientists posted dissertations and dusty data for reading with clunky, text-only browsers. With the advent of graphical browsers, the consumer-oriented Web took off. Content became vastly more colorful. Remember where you were the first time you experienced the exciting <blink> and <marquee> tags? (I bet you wish you could forget those gems!) Anyway, the Web has evolved as a rich, interactive, and personalized medium.

In the new version of Web (Web 2.0), functional pages aren’t enough. User experience (abbreviated as UX in geekspeak) is hot, and sites are cool. This chapter looks at Microsoft’s tools and technologies for creating and delivering engaging Web content.

Introducing the Content-Creation Tools

Microsoft has a range of tools for authoring Web pages that appeal to several skill levels. Some tools are more suited to Web page design, while others are more appropriate to programming.
Microsoft Office (Including Word 2007)

When Bill Gates realized that Microsoft was lagging on the Internet front, the word went out to integrate Web support into every product. As a result, you can save Excel spreadsheets, Word documents, and PowerPoint slides as Web pages.

Many companies use the Office suite to place information on their intranet because most employees are comfortable in Word and Excel. These tools are quite adequate for creating static Web content that some call brochure ware. Although somewhat bloated, the pages are faithful reproductions of the original document — especially when viewed in Microsoft’s latest Internet Explorer browser.

There’s nothing to stop you from using a “saved-as HTML” page in an ASP.NET site. However, you may find that removing the unwanted HTML markup takes more time than building the page from scratch.

Expression Web

Expression Web took over from Microsoft FrontPage as the content editor for professional designers. Although some see Expression as an advanced word processor for HTML pages, it’s actually much more, thanks to many important tools for Web designers. These tools include file management, link checking, style editing, and drag-and-drop support for HTML and ASP.NET controls.

Expression Web inherited the excellent split-view editor from FrontPage that lets you work in graphical and source code modes at the same time. The feature is so well done that Microsoft yanked the HTML editor from Visual Web Developer and substituted the superior Expression/FrontPage version.

Expression Blend

Expression Blend is mainly for the ponytail set (artistic types who prefer Macs) to create vector-based, animated, and three-dimensional graphics — much the way they do in Photoshop. Blend has a rich set of brushes, palettes, paint buckets, text, gradients, timelines, and event triggers for those with the skill to take advantage of them.

The XML-based files that Blend generates work in Windows Presentation Foundation (WPF) applications that run on Windows and in cross-platform Silverlight apps for the Web. (For more on Silverlight, see the section later in this chapter.)
Blend’s user interface (UI) is dim and funereal — a far cry from the cheerful Windows XP or glitzy Windows Vista UI. The theory is that a drab, flat design environment doesn’t distract an artiste from his or her canvas.

**Visual Web Developer (Including Express)**

Visual Web Developer (VWD) is the premier tool for programming Web sites on the Microsoft platform. Just as Word is part of the Office suite, VWD is part of the bigger Visual Studio 2008 suite. Visual Studio includes Visual Basic .NET, Visual C#, and many other tools. Visual Studio comes in several versions to target teams of developers, database designers, testers, and system architects.

As an integrated development environment (IDE), Visual Web Developer helps you assemble and build the key elements of a Web application, including Web pages, images, controls, databases, style sheets, and, of course, the programming logic.

Visual Web Developer Express (VWDE), shown in Figure 1-1, is a somewhat stripped-down, freebie version intended for beginners and hobbyists. VWDE doesn’t support add-ons, source control, extensibility, or macros — features that professional developers expect in a tool.

Most of this book’s instructions are common to VWDE and VWD. You can do almost everything in this book with the free Express product. I note the few places in the book (mostly when debugging) that apply only to the upscale ($$$) version of product. Chapter 3 gives you the cook’s tour of VWD.
Meeting the Technologies behind Web Applications

The technologies that support Web applications come from different organizations and from different teams within Microsoft. Here’s an overview of the parts that plug into — or on top of — each other.

Microsoft’s .NET 3.5 Framework

The .NET Framework is the base of what geeks call the stack.

You can think of the stack as a multilayered wedding cake where layers depend on the layer below for support. The .NET Framework (technically, a compiled portion called the Common Language Runtime, or CLR) sits at the bottom, and its code talks to the underlying operating system, such as Windows Server 2008 and Windows Vista. ASP.NET 3.5 depends on the .NET 3.5 Framework. (See the next section for more on this framework.)

You hear geeks refer to classes or class libraries that make up the .NET Framework. They use dot-filled names like System.Web, System.Data, and System.Xml.Linq. This dotty stuff is just a way to organize and categorize thousands of chunks of prewritten code that programmers can tap into via programming languages, such as C#, C++, and Visual Basic.

Microsoft provides tons of reference documentation on everything that’s in the .NET Framework. If you still don’t find what you need, you can peek into its source code to see how Microsoft makes it all work.

ASP.NET 3.5

ASP.NET 3.5 is a technology to deliver interactive, data-driven Web applications over the Internet and intranets. ASP.NET includes a large number of prebuilt controls, such as text boxes, buttons, images, and data grids, that you can assemble, configure, and manipulate with code to create HTML pages that correctly appear in all popular browsers.

When combined with programming logic, ASP.NET lets you send HTML code that’s specific to each user’s circumstances or requests. For example, if a user wants a Web page to show HTML tables with green text and a purple background, your code can read the incoming request, verify that it’s doable, and respond. This ability to create personalized, custom pages is known in the business as creating content on the fly and is a hallmark of server-side Web applications. Given that most people don’t want green text on a purple background, the “special-orders-don’t-upset-us” flexibility becomes a real bonus.
Unlike static HTML pages that are stored on disk in a fully complete state, ASP.NET pages usually exist in a skeleton-like state on disk. It’s only when a user requests a page that ASP.NET analyzes the markup, fills in all the content (often from a database), and sends HTML that the browser can render.

That’s a very quick summary of what ASP.NET does. Don’t fret if you don’t grasp it all yet. You can fill in the blanks as you jump around the rest of the book.

**ASP.NET Futures**

The ASP.NET Futures releases consist of controls and technologies that the ASP.NET team is tinkering with or would like to demonstrate. It’s a way of getting feedback, testing scenarios, and pushing the envelope without making a commitment to release the product.

The Futures items have no official support, even though some work quite well. Some components, such as the dynamic data controls, get their start in ASP.NET’s Futures farm team and end up as professionals in an ASP.NET release or extensions update.

**ASP.NET 3.5 Extensions**

The ASP.NET team continues adding controls between official releases. These are packaged as extensions that you can download and install. As of this writing, the ASP.NET 3.5 Extensions include the Silverlight and MediaPlayer controls for presenting rich media on ASP.NET pages. Other
recent extensions and templates include Dynamic Data controls for displaying database content and an advanced architectural framework called Model View Controller (MVC).

Microsoft has many terms for unfinished software such as alpha, beta, preview, community technical preview (CTP), and release candidate. For critical production use, check whether an ASP.NET extension has made it to the Released to Web (RTW) or Released to Manufacturing (RTM) stage.

**Web services**

Web services let you deliver data and calculations to remote computers without restricting your client base to those running Windows. The most popular exchange format is the Simple Object Access Protocol (SOAP), which lets different platforms talk to each other by using XML.

Microsoft put a big push into Web services via ASP.NET in previous .NET releases. The follow-on emphasis has been on services using Windows Communication Foundation (WCF). WCF services are more robust and easier to secure, especially for enterprise applications where you may be sharing healthcare data with a company that handles the billing.

Smaller Web sites also have some interesting uses for services, especially when hooked in with technologies such as ASP.NET AJAX. See Chapters 9 and 15 for examples of Web services.

**JavaScript and client-side code**

Modern browsers understand an internal programming language called JavaScript. When the browser encounters JavaScript code (*script* in geekspeak) inside an HTML page, it runs the program’s instructions. The browser (the client) doesn’t need a connection to the server to run JavaScript code — it’s completely independent. Client-side script uses the processing power of the computer on which the browser is running. That’s a tremendous advantage because it takes the pressure off the Web server and distributes tasks to individuals.

Client-side scripting becomes complicated — and extremely powerful — when combined with logic on the server. Imagine this scenario: The Web server sends a stream of HTML that contains JavaScript instructions. Those instructions include JavaScript code that checks whether the anonymous user has typed a number from 1 to 10 in a text box. The browser sees the script and executes it locally. Until the user has typed a number from 1 to 10, the Web server isn’t involved. When the browser sends the number back to the Web server, the return action is known as a *postback*. (See the sidebar “Postbacks and the rural mail carrier.”)
The powerful part is that the logic on the server can determine that 20 is an acceptable maximum number for a different customer and send a 20 in the JavaScript rather than the value 10. This way, the server is creating customized, client-side JavaScript on the fly.

**ASP.NET AJAX**

Asynchronous JavaScript and XML (AJAX) is a technology that reduces unnecessary and wasteful full page refreshes by limited the transfer of data to and from the Web server. (See the sidebar “Demolishing the house to change a window.”)

On an AJAX-enabled page, you can type your credit card number in a text box, click the Submit button, and get a response such as “Credit Card Accepted” without disrupting the images, menus, and text elsewhere on the page. The browser sends only the required data to the server. When the message comes back, AJAX uses JavaScript code and Dynamic HTML to write into the designated part of the page.
Microsoft’s flavor of AJAX is an integral part of ASP.NET 3.5 rather than an add-on as in previous releases. As a result, if a bug or security flaw exists, Microsoft can fix its AJAX code via Automatic Updates or during the monthly celebration known as “Patch Tuesday.”

You see AJAX in action throughout this book, but specifically in Chapters 4 and 15.

**Dynamic HTML**

While not exclusively a Microsoft technology, Dynamic HTML (DHTML) plays an important role in making Web pages responsive, interactive, and more like a regular Windows program.

When the browser analyzes the HTML code for a page, it creates an in-memory document. This document has a hierarchical structure where child elements nest inside their parent containers. For example, table rows are nested inside tables that are nested within the document’s body.

The word *dynamic* in DHTML refers to the ability to change the characteristics of an element by using JavaScript. You’ve seen this ability many times without necessarily paying attention. For example, you’re seeing DHTML at work when you hover the mouse over an image, and the image changes. Likewise, DHTML is at work when you click a plus sign to expand a paragraph of text. Chances are, JavaScript is instructing the text (or its container) to become visible — even though the original code sent from the server set the text as hidden.
The ability of JavaScript and ASP.NET AJAX to manipulate and rewrite almost any part of a Web page (the text included) is what makes most dynamic effects possible.

**Extensible Markup Language (XML)**

Although Microsoft had a hand in the specifications for Extensible Markup Language (XML), the standards come from the World Wide Web Consortium (W3C). Microsoft uses XML extensively in its Web technologies as a way of passing data around. These data exchanges include browser-to-server, server-to-browser, server-to-server, and from one program to another. You see XML in Chapter 7 as part of LINQ to XML and again in Chapter 9 within Web services. XML is also a big part of AJAX.

XML data has three big advantages:

- It’s generated as plain text so that it passes easily through firewalls.
- Humans can read it and make at least some sense of it.
- You can create, parse, and manipulate XML on any platform, not just on Microsoft’s operating systems.

**Silverlight**

Silverlight is Microsoft’s cross-browser, cross-platform multimedia plug-in. It works on Windows, Macs, and even the rival Linux platform.

You’ve almost certainly seen Macromedia (now Adobe) Flash movies on a Web page. Silverlight is like Flash, only faster, more technologically advanced, and easier to program, especially in .NET languages. This so-called Flash killer uses a form of XML markup called XAML (sounds like zamel and rhymes with camel) to generate its graphics and behaviors.

You can use Silverlight, shown in Figure 1-2, to embed everything from screencams to animated cartoons to full-motion video using live, streaming broadcasts. The download size is reasonable, and Silverlight runs in its own isolated area, known as a sandbox, so the program should be secure enough for most uses.
Silverlight is very appealing as a multimedia platform. It promises to be a very big deal as the tools and technologies become more advanced. Expect to see entire database-driven applications running on Silverlight that maintain their appearance even when you resize the browser. You can dip into Silverlight and other rich media types in Chapter 16.

**Language Integrated Queries (LINQ)**

Language Integrated Query (LINQ) is a set of additions to the C# and VB.NET programming languages that make it easier to deal with data. LINQ comes in several dialects, including LINQ to SQL, LINQ to XML, and LINQ to objects. After you master LINQ’s statements and syntax, you can apply the knowledge to all sorts of data. In fact, LINQ lets you combine data from multiple sources, such as a database, Web service, and XML file.

For most people, the big payoff is LINQ’s support for SQL Server. Instead of writing complicated SQL statements — and crossing your fingers that no syntax errors occur — LINQ lets you use familiar keywords in queries. Visual Web Developer (as with other members of the Visual Studio 2008 family) watches what you type and alerts you to problems.

Chapter 7 shows how to use LINQ to select, sort, and group data of all kinds. Chapter 8 focuses on the LinqDataSource control and DataContext object in ASP.NET applications and shows how to massage SQL Server data by using LINQ to SQL.
ADO.NET

ADO.NET is Microsoft’s technology for working with data and databases of all types. When a Web application talks to a database such as Microsoft SQL Server, it’s probably using ADO.NET. The introduction of LINQ has hidden much of ADO.NET from view in Visual Web Developer.

SQL Server

SQL Server 2005 and 2008 are key products in Microsoft’s Web technology strategy. The phrase “It’s all about the data” applies to most serious Web applications. Whether you're tracking user preferences, generating complex reports, or storing customer orders, you need a fast and reliable data engine and relational database.

Microsoft provides SQL Server Express for free (but, as they say, “connect charges may apply”), making it a great choice for beginners. The skills and data you acquire by using SQL Express are directly transferable to the latest versions of SQL Server from standard to enterprise. You use SQL Server (mostly the Express version) throughout the book.

Internet Information Services

Internet Information Services (IIS) is Microsoft’s premier Web server product that comes free with the latest versions of Windows.

As a platform, IIS delivers Web pages and Web services as requested by a browser or other application. ASP.NET 3.5 meshes seamlessly with IIS to produce the dynamic pages you’re reading about in this chapter.

You can run IIS on your developer workstation, over your company’s intranet, or expose it to the vast public on the Internet. However, unless you’re running a large business on the Internet, you probably use IIS through an independent hosting company. These hosters are specialists who rent space on their servers, sell bandwidth, maintain connections to the Internet, and schedule backups.

During the development stage in Visual Web Developer, you may not use IIS at all. VWD includes a light Web server that does almost everything you need on your local development machine. When you’re satisfied with the pages and code, you transfer the site to an IIS machine from within VWD. (For details on deployment, see Chapter 20.)
Chapter 2

Getting Up and Running

In This Chapter

- Installing Visual Web Developer Express
- Setting up the development environment
- Managing the Toolbox
- Using Solution Explorer and the Properties window

Technically, you don’t need Visual Web Developer Express to create Web pages for ASP.NET. All the source and configuration files are text-based combinations of HTML, XML, and computer code. You could just fire up Notepad and start typing, although, it would take days to create anything worthwhile.

In reality, you need an integrated development environment (IDE) to automate the creation of files, generate the code, organize the content, and keep all the tools in one place.

This chapter brings you up to speed on Visual Web Developer Express as the design environment for ASP.NET pages. This chapter doesn’t cover everything the IDE can accomplish for you because you’re itching to build pages. When you understand the basics, you can get a grip on the rest of tools in subsequent chapters.

Installing Visual Web Developer Express

This section takes you through the installation of Visual Web Developer 2008 Express Edition (VWDE). If you’re using the full Visual Web Developer product that’s part of Visual Studio 2008, your setup is somewhat more involved because you have more choices. Apart from the installation, everything in this book about the Express edition applies to the paid version of Visual Web Developer. I flag instances where a feature’s available in the paid version but not in VWDE.
If you’ve installed prelease versions of VWDE, Visual Studio 2008/Orcas, SQL Server, SQL Server Express, or the .NET Framework 3.5, I recommend you uninstall them to start with as clean a system as possible. Always use the Windows uninstall utility to remove software. In XP, choose Control Panel➪Add or Remove Programs; in Vista, choose Control Panel➪Programs and Features.

You can download a free copy of Visual Web Developer 2008 Express Edition from Microsoft’s Web site. The full installation described in this section requires 3.5GB on the C: drive and a total download of 447MB.

If part of the installation fails, return to the Web site shown in the first step and begin the process again. The installer should pick up where it left off.

Follow these instructions to download and install VWDE:

1. **In Internet Explorer, browse to**
   http://www.microsoft.com/express/download/.

2. **In the Web Install area of the page, locate the Visual Web Developer 2008 Express Edition product section, as shown in Figure 2-1.**

3. **Click the Download link.**
   A warning prompt appears, as shown in Figure 2-2.
4. Click Run.

The download of an initial 2.6 megabyte file starts.

5. If you’re using Windows Vista, click Continue on the User Account Control (UAC) warning, as shown in Figure 2-3.

6. On the Welcome to Setup screen (shown in Figure 2-4), click Next.

Sending anonymous setup information helps Microsoft analyze trends and catch oddball installation problems.

7. On the License Terms screen (shown in Figure 2-5), read every word of the license, consult your lawyer, select the radio button to acknowledge that you’ve read the terms (only if you read them!), and then click Next.
8. On the Installation Options screen (see Figure 2-6), select all the optional products.

You can leave out the MSDN Express Library to save time and bandwidth.

9. On the Destination Folder screen (shown in Figure 2-7), you can accept the default install folder (unless you have reason to change it) and then click Install.

Coffee break time! The full download and installation can take over an hour even with a fast Internet connection.
10. When the Setup Complete screen finally appears (see Figure 2-8), click Exit and, when prompted, click Restart Now to reboot the computer.

You have 30 days before VWDE nags you into registering the product.
Finally! Creating an ASP.NET Web Page

At this point, you have the software installed and you’re ready to take it for a spin. In this section, you create a trivial page so you can explore the environment. If you’re impatient to take on a larger project, jump to Chapter 3 and then return to “Tweaking Your Development Environment” later in this chapter for some configuration tips.

Starting the IDE

Similar to most installers, VWDE adds links to the Windows menu. To run the integrated development environment (IDE), click the Visual Web Developer 2008 Express Edition link from the main menu. The splash screen appears and the IDE comes alive. This can take a few moments (especially the first time) because there are background files to create and settings to write.

Okay! You’re viewing the default environment so prepare to make something happen!

Creating an ASP.NET Web site

Although you can edit a single Web page in VWD, you usually work on pages as part of a site. To create an ASP.NET Web site, follow these steps:

1. Choose File➪New Web Site.
   The New Web Site dialog box appears.

2. In the Templates section, near the top, select the ASP.NET Web Site template, as shown in Figure 2-9.

![Figure 2-9: Selecting the ASP.NET Web Site template.](image)
3. Ensure that File System is selected in the Location drop-down list.

4. Make sure Visual Basic is selected in the Language drop-down list.

5. Click OK.

The IDE goes to work and creates the Web site, a starter Web page, and some other files.

As shown in Figure 2-10, the IDE has three significant areas:

- The Toolbox and other tabs scrunched along the border on the left and easy to miss. The Toolbox glides into view when you pass the mouse over its tab. Try it!
- The default.aspx page opens in the Web page editor (the center).
- Groups of small windows on the right, including one titled Solution Explorer.

The Web page editor offers three views:

- **Design view:** Gives you a graphical page view that somewhat resembles what users see in the browser
- **Source view:** Shows you the source code for the page, including the HTML markup.
- **Split view:** Shows the graphical page view in one pane and the source code in another. When you make changes in one view, it prompts you to refresh the view.

![Figure 2-10: The Toolbox tab, editing area, and windows in the VWDE IDE.](image)
Adding an ASP.NET control

You can spice up the sad blank Web page by adding a control to it. A control is an object on a page that renders code or markup that a browser understands. Controls such as text labels, drop-down lists, grids, and text boxes are the objects that make Web pages interesting, dynamic, and useful.

To add a control to your Web page in Design view, follow these steps:

1. In the lower area of the page editor, click the Design button.
2. In the upper area of the editor pane, locate the default faint blue or gray box with dotted lines.
3. Put your mouse cursor into the inner rectangular area.
   The area (a visual representation of an HTML <div> tag) becomes more prominent and displays a tab.
4. From the Standard section of the Toolbox, drag a Button control and drop it inside the rectangle that you located in the previous step. (If the Toolbox isn’t showing, make it visible by choosing View ➪ Toolbox.)
   Figure 2-11 shows the Button control in Design view.

Admittedly, this ASP.NET page is skimpy, but it’s enough for this preliminary exploration of the IDE.

Previewing a page in the browser

You can browse a page even while it’s under construction if you’re inside Visual Web Developer. The environment provides a Web server to compile the ASP.NET source code and render the HTML. It also launches the browser for you. To browse to a page within the IDE, follow these steps:

1. Place the mouse inside the ASP.NET page that you’re editing.
2. Right-click and from the context menu, choose View in Browser.
If you haven’t saved your work, the IDE prompts you to do so. If prompted, click Yes.

Internet Explorer opens and displays your page. You might need to deal with some nuisance security alerts from the browser by clicking to enable intranet settings.

3. **For additional excitement, click the button on the page.**

Other than a minor page flash, nothing much happens.

4. **Close the browser.**

You can save, build, and browse by pressing Ctrl+F5.

---

**Tweaking Your Development Environment**

In the preceding section, you used the development environment with its default settings. Microsoft’s choices aren’t always the best, so the first thing you want to do is configure the IDE for ease of use.

### Showing all settings

For some reason, the people who created the IDE shield us from many customization features. To make sure all the settings are available, follow these steps:

1. **In Visual Web Developer Express, choose Tools ▸ Options.**
2. **In the bottom left corner of the Options window, check Show All Settings box.**

   Presto! You go from a few measly settings to more options than you can change in a day!

### Unhiding advanced members

Even though you’ve expanded your options dramatically by showing all settings, Microsoft is still holding back. When you’re working with automatic statement completion (IntelliSense), the default settings hide many statements. To unhide the advanced members, follow these steps:

1. **Open the Options window (Tools ▸ Options).**
2. **Expand the Text Editor node and select All Languages.**
3. **In the Statement Completion area, clear Hide Advanced Members box.**
Before you close the Options window, you may want to deselect Enable Single-click URL Navigation. The “feature” adds distracting hyperlinks to any text that resembles a Web address.

Starting pages in Design view

If you’re working mainly with graphical design tools it’s more convenient to open ASP.NET pages in Design view. Here are the steps to configure Design view as the default:

1. Choose Tools ➪ Options.
2. If you haven’t already done so, check the Show All Settings box.
3. Expand the HTML Designer node and click the General node.
4. In the upper area of the window, in the Start Pages In group, select Design View.
5. Click OK.

Working with the Toolbox

The Toolbox is where you store, um, tools. You use the Toolbox frequently in this book, so it helps to know its tricks.

Auto Hide and the pushpin

By default, the Toolbox plays peeka-boo to give you the maximum screen real estate as you work in the page editor. If you want the Toolbox to remain visible, pass your mouse over the Toolbox tab and click the pushpin so the pushpin becomes vertical, as shown in Figure 2-12.
If the Toolbox disappears completely, choose View ➪ Toolbox.

**Adding controls to the VWDE Toolbox**

Visual Web Developer Express doesn’t support higher-end automation features that let third-party installers fill the Toolbox for you. You can still use controls that you build or buy, but you need to add components manually. As you see in the following sections, adding stuff to the Toolbox is no big deal.

**Obtaining the AJAX Control Toolkit**

This section shows how to add controls to the Toolbox by using the free AJAX Control Toolkit as an example. These same steps apply to other non-Microsoft controls but because I show you how to use the AJAX Control Toolkit extensively in Chapter 15, it’s handy to have it ready.

Here’s how to obtain a copy of the Toolkit and prepare it for installation:

1. **Browse to the following URL:**
   - www.codeplex.com/AtlasControlToolkit

2. **Click the Releases tab.**

3. **Click the link to download the latest NoSource Zip file (probably AjaxControlToolkit-Framework3.5-NoSource.zip) that contains the runtime binary file you want. You must agree to the license.**

4. **After the file is downloaded, right-click the file in Windows Explorer and select Extract All.**

5. **For the destination folder, type c:\ACT\ and click Extract.**

   It might take a few seconds to extract the files.

**Putting the AJAX Controls into the Toolbox**

Follow these steps to create an AJAX Control Toolkit tab and add its controls to the Toolbox:

1. **Create a new Web Site (File ➪ New Web Site).**

2. **In the Toolbox, scroll to the bottom of the window and right-click the blank area below the General group.**

3. **From the context menu, choose Add Tab.**

   A blank group appears.

4. **Enter the group name, AJAX Control Toolkit.**

   A message indicates that no usable controls are in the group.
5. Right-click the area below the group name and from the context menu, select Choose Items.

   The Choose Toolbox Items dialog box appears.

6. Click the Browse button in the lower right area, and browse to c:\ACT\SampleWebSite\Bin\.

7. Select AjaxControlToolkit.dll (the Toolkit’s binary file) and click Open.

   The controls appear with check marks in the Choose Toolbox Items dialog box.

8. Click OK.

   The control names and their icons appear below the tab, as shown in Figure 2-13.

---

**Figure 2-13:** Controls from the AJAX Controls Toolkit in the Toolbar.

---

**Tip**

Double-click a control name in the Toolbox to add the control to the current ASP.NET page.

---

**Peering into a Wall of Windows**

Visual Web Developer has scads of work areas that you need to explore at some point. Given this book’s “just-in-time” philosophy, here’s a look at two windows that you use frequently while creating pages: Solution Explorer and the Properties window.

---

**Organizing files with Solution Explorer**

Solution Explorer is where you add files and folders to your Web application, much like you can in Windows Explorer. Figure 2-14 shows how Solution Explorer looks when you start a new Web site.
Visual Web Developer Express doesn’t have Solutions but it has a Solution Explorer. Why? Well, the term Solution comes from other Visual Studio products where a Solution acts like a master project.

You can drag files from your file system and drop them into Solution Explorer, but it’s more common to add pages, style sheets, files, and folders based on preconfigured starter files called templates. Follow these steps to add an XML file to the project from a template:

1. In Solution Explorer, just under the top row of icons, right-click the project name (which probably looks like C:\...\Website1\).

2. From the context menu, choose Add New.
   The Add New Item dialog box opens with a list of installed templates.

3. From the list of templates, select XML File, and then click Add.
   The starter XML file opens in the editor.

You’re not going to do anything with this XML file. You can delete it if you want by pressing the Delete key.

Whenever you need a command and don’t know where to look, right-click. Chances are the command is sitting on the context menu.

**Setting Properties in the Properties window**

When geeks talk about properties, they’re not discussing real estate. They’re referring to an item’s characteristics. For example, if you said, “I want a clear day for our picnic,” a weather-oriented geek would configure the sky like this:

```csharp
Sky.Visibility = Clear
```
In effect, the geek is setting the sky’s Visibility property to the value of Clear.

You encounter property/value pairs frequently in VWD. You configure almost everything by using properties and values. Designers and wizards that are built into the IDE configure properties for you. Follow these steps to open a Properties window and set the values for a Button control:

1. Open an ASP.NET page in Design view.
2. From the Toolbox, drag a Button control from the Standard category, and then drop it on the page’s design surface.
3. Select the button you just added.
   By default, properties sort in categories, such as Accessibility and Appearance.
5. Switch to Alphabetical view by clicking the AZ icon in the upper area of the Properties window.
6. Locate the (ID) property (shown highlighted in Figure 2-15) and change the value from Button1 to btnAccept.
7. Click the BackColor property.
   An ellipsis button (...) appears in the right-hand column.
8. Click the ellipsis button (...) to open the Color Picker dialog box.
9. In the Color Picker, select a color and then click OK.

Now that you know what the Properties window is about, you can expect to see instructions in a shorter form. For example, Table 2-1 shows how books often present property values for configuration.
The first row of data in Table 2-1 means to open the Properties page for the control, look for the property called ID, and set its value to btnAccept. You set some values with interesting secondary windows such as the Color Picker, and other values by choosing from a drop-down list. Much of the time, you just type the value (without quotations).

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>btnAccept</td>
</tr>
<tr>
<td>BackColor</td>
<td>#CCFFFF</td>
</tr>
<tr>
<td>Text</td>
<td>Accept</td>
</tr>
<tr>
<td>UseSubmitBehavior</td>
<td>False</td>
</tr>
</tbody>
</table>

**Table 2-1 Sample Button Control Values**

*Viewing what the Properties window has generated*

Properties windows and other designers write lots of accurate code quickly. There's no mystery to what's going on because you can view the generated code at any time. Follow these steps to see the results of the settings from Table 2-1:

1. **Locate the page that contains the button you configured in the previous section of this chapter.**
2. **At the bottom of the editing area, click the Source button.**

   The source code for the page appears.

3. **Locate the source code for the ASP.NET Button control.**

   For your reference, the reformatted markup is:

   ```xml
   <asp:Button
       ID="btnAccept" runat="server" Text="Accept"
       BackColor="#CCFFFF" UseSubmitBehavior="False" />
   ```

   The Properties window wrote the property names and values for you. The values are in quotes because ASP.NET markup follows XML syntax rules, which require wrapping attribute values in quotation marks.
In This Chapter

- Creating a Web project
- Creating and using a SQL Server Express database
- Generating a Web page based on a database

My niece Julie is always adding to her DVD collection. It’s hard to keep track of what films she owns. A good solution — and a great project for this chapter — is a Web page where Julie can log and display her latest acquisitions.

Most ASP.NET Web applications revolve around data and this one’s no exception. This chapter introduces you to Microsoft’s SQL Server database where you store the DVD information. Data isn’t much use if you can’t display it, and that’s where the ASP.NET data controls enter the picture.

The striking feature of this Web application is the way you create it without writing any code. Don’t get me wrong, there’s code in the app; it’s just that you let Visual Web Developer (VWD) write it according to your instructions.

Creating the DVD Web Project

In this section, you create a Web site project, add files and folders to it, and build the pages.

This chapter assumes that you installed and tweaked your environment as described in Chapter 2. If something seems to be missing (like SQL Server Express), refer to the preceding chapter for the installation and configuration instructions.

Follow these steps to create the JulieDVD Web project:

1. In Visual Web Developer, choose File→New Web Site.
2. In the New Web Site dialog box, select the Empty Web Site template, as shown in Figure 3-1.

This time, you start the site with no files and build everything from scratch.


4. In the Location box on the right, type the name of a new directory (for example, c:\JulieDVD, as used in Figure 3-1) or browse to an existing directory to store the project.

5. Click OK.

Don’t include spaces or punctuation in folder or filenames in Web applications. Although the Web server and browser might allow them, files with nonalphabetic characters can be hard to use.

Using a SQL Server Express Database

You won’t get far in building data-driven Web applications without a database. In this section, you work within the Visual Web Developer environment to create, configure, and enter data in a database table.

Adding a database to the project

This section assumes that you installed SQL Server 2005 Express (or a newer version) on your development machine. You can run the installer again to add options. To add a SQL Server Express database to your Web project, follow these steps:
1. In Solution Explorer, right-click the App_Data folder (created in the previous section), and then click Add New Item.

2. In the Add New Item window, select SQL Database.

3. Change the name of the database to JulieDVD.mdf and then click Add.

   VWD warns you that you should place a database in the special App_Data folder. ASP.NET provides appropriate security permissions for the App_Data folder.

4. Click Yes to place the database in the App_Data folder.

   The IDE goes to work, generating an empty database.

   The Database/Server Explorer window appears in the IDE. Read on to use it.

### Adding a table to the database

Databases store data in tables. It’s logical, then, that a database without a table isn’t very useful.

To add and configure a database table, bring up Database/Server Explorer (View ➤ Database/Server Explorer), right-click the Tables node and, from the context menu, select Add New Table. As shown in Figure 3-2, Table Designer appears. This is where you configure table columns.

![Figure 3-2: Table Designer for configuring data tables.](image)

#### Adding an identity column in Table Designer

For Julie’s DVD page, you want to store the following information:
The title of the movie
A description of the movie
The date she added the movie to her collection
A unique identifier (or identity column)

The first three items in the list are obvious, because that’s information that you want to add and use. The last item, an ID, is mainly for the database itself. An *identity column* helps the database track data more efficiently.

In the following steps, you add a column (also known as a *field*) to the table and instruct the database to assign the ID numbers automatically:

1. **In the blank space below the Column Name heading, type ID.**
2. **In the area below Data Type, type int.**
   By setting the Data Type to int, you’re telling the database to expect ordinary numbers like 4 or 99 in this column.
3. **Underneath Allow Nulls, clear the check box.**
   *Null* is a geeky term for *absolutely nothing*, *rien*, *zilch*, and *nada*. When you uncheck Allow Nulls, you’re telling the database to report an error when a program fails to put something into the column.
4. **On the Column Properties tab, expand the Identity Specification node, as shown in Figure 3-3, and set the (Is Identity) value to Yes.**

5. **Select the ID column and, from the Table Designer menu, choose Set Primary Key.**
   A little key icon appears next to the column name.
That’s all you need to do to make every record in the table unique. The database supplies the number when you add a record.

**Adding regular columns to a table**

In this section, you configure the columns that hold the data you really care about, such as the DVD’s title and description. Here are the steps to add regular columns to the database table:

1. **In the empty row in Table Designer (underneath the ID row), type Title in the Column Name column.**
2. **For the Data Type, enter varchar(50).**
   
   A varchar type holds text characters like the ones you find in a movie title; in this case, it holds 50 characters.
3. **Clear the Allow Nulls check box.**
   
   This makes the movie’s title a required field when adding a movie to the database.
4. **Underneath the Title row, add a row called Description with the Data Type varchar(500), but leave Allow Nulls checked.**
   
   This time, you have lots of space (500 characters) to describe the movie. However, movies don’t need a description, so Null is allowed in this case.
5. **Add a column called DateAdded (no space!) with the Data Type datetime, and uncheck Allow Nulls.**
   
   Your Movies table now has four columns, as shown in Figure 3-4.

6. **Close the Table Definition pane by clicking the X at the upper right.**
   
   The IDE prompts you for a name for the table.
7. **Type Movies as the table name, and then click OK.**
Adding data to a table in Query Designer

A database is much more interesting with real data in it. You don’t have a Web page yet to enter data into the database, so this is a good place to show you how to work directly in the database table with Query Designer. Follow these steps to add rows of data to the Movies table in Julie’s DVD database:

1. Open Database/Server Explorer (View ➪ Database/Server Explorer) and navigate to the node for the Movies table (Data Connections ➪ JulieDVD.mdf ➪ Tables ➪ Movies).

2. Right-click the Movies node, and choose Show Table Data from the context menu that appears.

The table opens in VWD’s Query Designer, as shown in Figure 3-5. There’s nothing much to see except the column names and the geeky keywords NULL.

3. In the Title column, put your cursor on NULL and type Italian Job.

4. Tab to the Description column and type To be provided.

The red exclamation mark warns that what you’ve typed so far isn’t saved.

5. Tab to the DateAdded column and enter a full date, such as November 11, 2007.

Query Designer has an automatic date conversion that is handy when you’re not sure whether the database expects the order as day/month/year (British and Canadian) or month/day/year. Type it in English and let the tool figure it out!

6. Tab to the next row.

Notice that the database has provided the ID for the row you just created.

Query Designer enforces rules when you enter data. For example, it complains loudly if you don’t enter a date because NULL isn’t allowed in the DateAdded column.

You end up with a gridlike table resembling Figure 3-6. If you have many rows, you can use the VCR-like controls to navigate among them.
That’s all you need to do for the moment with the database. In the next section, you use the power of VWD’s designer tools to generate code.

**Generating a Data-Driven Web Page**

Did you notice that the heading says *Generating* rather than *Programming*? In this section, VWD generates tons of code as you drag and drop.

**Adding a single file model Web page**

First, you need to create a page so you have a place for the code it creates. For this simple project (and most of the examples in this book), you use the single file model where everything goes into the `.aspx` page. See the “Choosing single file model or code-behind” sidebar to decide which is right for you. To add a Web page that uses the single file model, follow these steps:

1. In Solution Explorer, right-click the project name, and then choose Add New Item.
2. In the Add New Item window, select the Web Form template.
3. Type `default.aspx` as the name of the file.
4. Clear the check box for Place Code in Separate File. This ensures that the IDE puts all code within the `.aspx` file.
5. Click Add.

The new, empty page appears in the IDE.

**Using the database to build a Web page**

Instead of adding controls to the page from the Toolbox, let the IDE do the work. You need a SQL Server database (Express is fine) for this magic. I show you how to create the database in the previous section, “Using a SQL Server...”
Express Database. To generate a user interface based on a database table, follow these steps:

1. **Make sure the Web page is open in Design view.**

2. **In Database/Server Explorer (View ➪ Database/Server Explorer), locate the Movies table that you created previously (Data Connections ➪ JulieDVD.mdf ➪ Tables ➪ Movies).**

3. **Drag the Movies table from Database Explorer and drop it inside the div block on the Web page.**

As shown in Figure 3-7, the IDE adds two controls to the page: GridView (the HTML tablere-like control) and SqlDataSource (represented by a gray block).

### Choosing single file model or code-behind

ASP.NET allows you to keep both the HTML and Visual Basic (or C#) source code in one file that uses the .aspx extension. The other option is to keep the markup in the .aspx file and put the source code in a second file. Geeks call the latter model code-behind because they consider the .aspx page as the face of the page and the code logic hanging around as the background.

In small projects, I prefer the single-file model because there’s one less file to track and deploy. The two-file model (using code-behind) appeals to computer science purists who like a physical separation of the presentation aspects (the HTML content) and the coding logic.
answer (no, no code yet) highlights the advantages of knowing your tools and letting them do the work.

**Previewing and reviewing the database-generated page**

The proof of any database-generated page is in the running. If you’ve been holding back from trying out the page, you have admirable self-control. Here’s how to give it a whirl in the browser:

1. Right-click the design surface of the page, and choose View in Browser from the context menu.
2. If you’re prompted to save changes before previewing the file, click Yes.

Figure 3-8 shows the browser preview of the page. It’s not bad considering the minor effort to create it. You enhance it in the following chapter.

---

**Figure 3-8:**
In the browser, the database-generated page displays all the columns found in the underlying table.
The tiny Web server

When you browse ASP.NET pages on your development computer, you sometimes see a notification balloon like the one in the following figure. This tells you that a tiny Web server is handling the server-side processing that ASP.NET requires.

Notice the number 49159 in the URL? That’s the random port on your machine that the development environment uses for the browser connection.
Chapter 4

Managing Data and Other CRUD

In This Chapter
- Using Tasks menus
- Inserting, sorting, editing, and deleting data
- Formatting a date
- Using the FormView control’s templates
- Improving performance with the UpdatePanel

The preceding chapter shows how to create a read-only list of DVDs by building a database and dropping a database table onto a Web page. Although it’s a great start, you haven’t met the requirements set by the client (my niece Julie). Julie wants CRUD on her page:

✔ **Create**: The ability to insert new DVDs into the database
✔ **Retrieve**: Fetch the list of DVDs and display it nicely
✔ **Update**: Change the information about a DVD in the collection
✔ **Delete**: Remove a DVD from the list

This chapter uses the same ASP.NET page and database as the last chapter. You can create the page and database by going through the steps, or you can shortcut the process by downloading everything from this book’s Web site.

Here are some other enhancements you supply in this chapter:

✔ Make the font and table design more attractive
✔ Fix the column title text
✔ Do something about the ridiculous, geeky time on the date display
✔ Replace *Untitled Page* with a real title
✔ Get rid of the screen flash
Working with Smart Tags and Designers

The more sophisticated ASP.NET controls, such as the GridView control, include designers. Designers are wizardlike functions that help you configure the control by making choices. Many of the designers appear on special Tasks menus that you reach by using a Smart Tag. A Smart Tag is a shortcut panel that pops up next to an ASP.NET control at design-time.

Showing the Smart Tag and tasks via a menu

One way to show a Smart Tag is to use the context menu. In an ASP.NET page, select any ASP.NET control and right-click. The Show Smart Tag item is enabled if the control supports the Tasks menu.

Using the Smart Tag button

A common way to open the tasks is to use the Smart Tag button. In Figure 4-1, the mouse pointer shows the tiny arrowlike button in the upper right.

Figure 4-1: The tiny arrow in the upper right of a design-time control is the Smart Tag button that opens a control's list of tasks.

<table>
<thead>
<tr>
<th>ID</th>
<th>Title</th>
<th>Description</th>
<th>DateAdded</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abc</td>
<td>abc</td>
<td>14/06/2007 12:00:00 AM</td>
</tr>
<tr>
<td>1</td>
<td>abc</td>
<td>abc</td>
<td>14/06/2007 12:00:00 AM</td>
</tr>
<tr>
<td>2</td>
<td>abc</td>
<td>abc</td>
<td>14/06/2007 12:00:00 AM</td>
</tr>
<tr>
<td>3</td>
<td>abc</td>
<td>abc</td>
<td>14/06/2007 12:00:00 AM</td>
</tr>
<tr>
<td>4</td>
<td>abc</td>
<td>abc</td>
<td>14/06/2007 12:00:00 AM</td>
</tr>
</tbody>
</table>

To make the arrow button visible, select the control first — not easy on a busy page. Then click the arrow to see whatever the Smart Tag has to offer.
Enhancing the GridView Control

You fulfilled the Retrieve requirement of the CRUD implementation at the end of the previous chapter. The GridView control presents the data in a rather ugly grid. You beautify it in the next sections.

Adding a dash of color to the GridView control

The drag-and-drop routine in the preceding chapter put a GridView control on the page. The GridView control is one of several powerful and versatile design-time controls that ship with ASP.NET.

Visual Web Developer supports the design-challenged by providing some reasonably attractive starter designs. Follow these steps to use the AutoFormat feature:

1. Select the GridView control, and then click its Smart Tag button.
   The Tasks menu opens.
2. Click AutoFormat.
   The AutoFormat window, shown in Figure 4-2, previews the available schemes.
3. Select a scheme (for example, Professional), and then click OK.

Figure 4-2: The AutoFormat tool applies a quick and not-too-shabby makeover for the design-challenged.
Sorting, editing, and deleting with the GridView

The GridView control has several excellent features that are inactive by default. Follow these steps to add sorting, editing, and deleting to the GridView control:

1. **Open the GridView control’s Tasks list by clicking the Smart Tag button in the upper right of the control.**
   
The GridView control’s Tasks list appears, as shown in Figure 4-3.

2. **Enable the check boxes for Sorting, Editing, and Deleting.**
   
   If the check boxes are grayed out, make sure that the ID column in your database is set as an identity column and as the primary key. (Refer to Chapter 3.)

**Figure 4-3:** The GridView control’s Tasks menu includes check boxes to switch on sorting, editing, and deleting data.

Testing editing

Browse to the page (Ctrl+F5) and click the Edit button to launch edit mode. The Title, Description, and DateAdded fields turn from read-only text into text boxes that accept input. You can click Update to apply your changes, or click Cancel to abandon your edits, as shown in Figure 4-4.

Did you notice that the ID field in Figure 4-4 isn’t editable? When you designed the database schema in Chapter 3, you told the database to assign ID numbers automatically and you made ID the primary key. The GridView control wisely prevents you from messing with ID values.
Column sorting of sorts

Hmmm . . . if you only have one row of data, it’s tough to tell if sorting is working. Normally, you click the links in the column headers to sort the column, as shown in Figure 4-5. Clicking the same link again reverses the sort. Maybe you could come back to play when there’s more data?

Not testing deleting

The tempting Delete link removes the row from the database. You don’t want to use it yet. Keep reading.

There’s no built-in safety net on that Delete link and your data is gone in an ohnosecond. Too late? You can go to Chapter 3 to see how to insert new data. Or, continue in this chapter to “Using the FormView control to insert a row.”

Ohnosecond

ohnosecond: The span of time between carrying out an action and realizing that you’ve just made a huge, irreversible mistake. For example, you ridicule your boss in an e-mail to colleagues and, an ohnosecond after clicking Send, you realize the boss was one of those in the address list. The scientific measurement gets its name from the sorrowful English exclamation, “Oh No!”
Formatting the date display

Your GridView control’s DateAdded column is weird. For one thing, it’s hard to read a date that looks meant for a computer:

12/24/2007 12:00:00 AM

For another, nobody cares about the time (down to the seconds, no less!). In addition, a space is missing in the title of the date’s column header.

To display the date in a friendlier format, follow these steps.

1. **Open the GridView control’s Tasks list by clicking the Smart Tag.**
2. **From the menu, choose Edit Columns.**
   The Fields window opens.
3. **Locate the Selected Fields (not the Available Fields) area in the lower left of the window and click DateAdded.**
   The window shows the properties for the DateAdded column, as shown in Figure 4-6.

   **Figure 4-6:** You can change the properties of a GridView control’s column, including the header text and column formatting.

4. In the BoundField Properties section, set the following properties and values:
The starting and ending characters in the DataFormatString value are braces ({}), not regular round brackets. If the formatting doesn't work, make sure that you set HtmlEncode to False.

5. Click OK.

When you browse to the page after the preceding changes, the dates look like they're suitable for human consumption.

**Introducing the FormView Control**

The preceding sections of this chapter demonstrate the easy implementation of RUD (Retrieve, Update, Delete), using the GridView control. For the adding (Creating in CRUD-parlance) part of the acronym, you use ASP.NET’s FormView control.

Oh, and have you written any code yet? Keep it that way!

**Adding a FormView control to the page**

The FormView control lets you display, update, and add data via drag and drop. In this procedure, you only need the FormView control’s ability to insert (that is, add) a row. **Remember:** You're still working on the ASP.NET page that you added in Chapter 3.

Follow these steps to add a FormView control to the page and set its data source:

1. From the Toolbox, in the Data category, drag a FormView control and drop it below the existing GridView control.

2. Click the Smart Tasks arrow thingy to open the FormView control’s Tasks menu.

3. Choose a data source (usually, SqlDataSource1) from the drop-down list, as shown in Figure 4-7.
As configured, the FormView control does more than you need on this page, such as displaying and updating DVD data. In the following section, you strip out the unwanted functions by digging into the FormView control’s templates.

**Changing the FormView control’s templates**

The FormView control is a shape-shifter in that it can take on a dramatically different appearance depending on its current *mode*. For example, one instant the FormView control looks like a read-only display of data and the next instant it’s showing text boxes so users can enter data. You implement the appearance of the modes via a template. See Chapter 13 for more discussion on using templates.

A template acts as a container to hold the markup (HTML-like code) and inner controls required for a given mode. The FormView control has these templates:

- **ItemTemplate**: Used when displaying data
- **EditItemTemplate**: For implementing data editing mode
- **InsertItemTemplate**: Holds the markup required for adding data
- **EmptyDataTemplate**: Used when there’s no data to display
- **HeaderItemTemplate**: Creates header content for all modes
- **FooterItemTemplate**: Creates footer content for all modes
- **PagerTemplate**: Generates navigation markup for paging through data

Currently, you want a tiny part of the ItemTemplate’s default content and most of the InsertItemTemplate’s content. The rest? You throw it away. Follow these steps to configure the FormView control that you added in the preceding section:
1. **Open the FormView control’s Tasks menu by clicking its Smart Tasks arrow.**

2. **Choose Edit Templates (at the bottom of the menu) to enter template editing mode.**
   The Template Editing Mode window appears, as shown in Figure 4-8.

3. **Click ItemTemplate to begin editing the ItemTemplate template.**
   Figure 4-9 shows that the FormView control adopts the ItemTemplate appearance when displaying data. Notice that the template box includes static text (ID: and Title:) as well as ASP.NET Label controls (IDLabel and TitleLabel).

   Depending on your IDE settings, you might not see the arrow symbols in Figure 4-9 that represent new lines, and the dots that indicate spaces.

4. **Leaving only the New hyperlink, carefully delete the remaining content within the ItemTemplate box.**
   One technique is to put your cursor to the left of the New hyperlink and press the Backspace key.

   When finished, you’ve stripped the ItemTemplate to a very small hyperlink.

5. **Open the Template Editing Mode window again and, from the dropdown list, choose the display for the EditItemTemplate mode.**
6. **Delete everything within the EditItemTemplate container.**

Your page uses the GridView control’s inline editing feature, so you can remove editing capability from the FormView control.

7. **Open the Template Editing window (yet again) and this time, select the display for the InsertItemTemplate mode.**

As shown in Figure 4-10, the InsertItemTemplate is the FormView control template you use for inserting data.

8. **Place your cursor between the letters e and A of DateAdded and add a space to make it two words.**

9. **Open the FormView control’s Template Editing window and click End Template Editing**

The FormView control is back to its natural state. However, the appearance has change dramatically because you removed so much of the template content. All that’s left in the default view is the New hyperlink.

**Using the FormView control to insert a row**

The best way to tell whether an ASP.NET page is going to work is to run it. Follow these steps to add a new DVD to Julie’s database:

1. **Browse to the page (Ctrl+F5).**

The GridView control appears with the data, as shown in Figure 4-11. Notice also the New hyperlink below the GridView control. That’s the FormView control.
2. Click the New link.

The FormView control, shown in Figure 4-12, morphs into a data input area with space for the Title, Description, and Date.

3. Type a title and description of a DVD in the appropriate spaces.

4. In the Date Added field, type a normal date, such as December 24, 2007.

5. Click Insert.

The title appears in the GridView control, and the FormView control returns to its normal mode.

Congratulations! You’ve added Create, the final piece of CRUD. Oh, and you can try sorting now.
Analyzing problems with the date input

Technically, the page works. However, the Date Added field has problems; an error in the date crashes the page. Want proof of the page's fragility? Click the New button to add a DVD to the list but rather than a real date, type some nonsense characters, such as `blahblah`, and click Insert.

Yikes! The page crashes and reports an error (an `exception` in geekspeak). ASP.NET tried to find a usable date value in `blahblah` and choked on it. Read on to fix the problem.

Validating the date input

One way to prevent the error is to make sure the date is recognizable and valid before sending it to the database. In this section, you use a `CustomValidator` control and (gasp!) a line of server-side code to keep bad dates from getting near SQL Server. To validate the date, follow these steps:

1. Click the FormView control’s Smart Tag to open the Tasks menu.
2. Choose Edit Templates.
3. From the drop-down list, choose the InsertItemTemplate.
4. From the Validation category of the Toolbox, drag a `CustomValidator` control and drop it below Date Added.
5. Set the following `CustomValidator` properties (F4):

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ControlToValidate</td>
<td>DateAddedTextBox</td>
</tr>
<tr>
<td>Display</td>
<td>Dynamic</td>
</tr>
<tr>
<td>Error Message</td>
<td>Not a valid date!</td>
</tr>
<tr>
<td>SetFocusOnError</td>
<td>True</td>
</tr>
<tr>
<td>ValidateEmptyText</td>
<td>True</td>
</tr>
</tbody>
</table>

6. Switch to Source view and put the cursor between the `<script runat="server">` and `</script>` tags.

7. In the Object drop-down list (top area of the source code frame), choose CustomValidator1, as shown in Figure 4-13.
8. From the Event drop-down list (to the right of the Object list), choose ServerValidate, as shown in Figure 4-14.

The IDE inserts a procedure that handles the CustomValidator1 ServerValidate event.

9. Insert the following line of code above the closing End Sub:

   ```vbnet
   args.IsValid = IsDate(args.Value)
   ```

   The code uses Visual Basic’s IsDate() function to determine whether the contents of the text box can be converted into a usable date.

Browse to the page and insert a bogus date. The CustomValidator is watching during the ServerValidate event.

Look for more detail on events in Chapter 5. For more on validation, see Chapter 19.

## Fixing the Page Title

It’s highly unlikely that anyone wants to call an ASP.NET page Untitled Page. Follow these steps to change the silly default title:

1. Open the ASP.NET page in Design view.
2. Click a blank area of the design surface, say, at the very bottom.
3. Open the Properties window (F4).
4. Change the document’s Title attribute to Julie’s DVDs.
Improving Performance with the AJAX Update Panel

Nobody likes to wait for a Web page to refresh. In this section, you tackle the evils of postbacks to improve performance. With ASP.NET AJAX, it takes only a couple of minutes to improve the feel of your page. Follow these steps to implement AJAX enhancements:

1. From the Toolbox, in the AJAX Extensions category, drag a `ScriptManager` control and drop it before any other controls at the top of the page.
2. From the Toolbox, drag an `UpdatePanel` control and drop it below the `ScriptManager` control.
3. Drag the `GridView` control and drop it inside the `UpdatePanel` control.
4. Drag the `FormView` control and drop it inside the `UpdatePanel` control.
5. Test the page to confirm that there’s no longer a full page-refresh when adding or editing an item.

At runtime, click the links and notice how smooth the page feels. For more ways to use AJAX, see Chapter 15.
Even in science fiction, you can’t escape manual data input. During an attack, spaceship navigators converse comfortably with computers, use console controls, and type quadrant coordinates.

This chapter looks at some key ASP.NET controls, forms, and events. Some concepts are easier to understand if you know a programming language; however, there’s no reason you can’t pick this stuff up while you go along.

**Accepting Data in a TextBox Control**

The ASP.NET TextBox control accepts keyboard input. As shown in Figure 5-1, the control appears as (depending on the TextMode property) a normal text box, a password variation, or a multiline version.

See Chapter 15 for enhancements to the TextBox control such as a prompting effect and masked input.
Creating a regular text box

You add an ASP.NET TextBox to your page by dragging it from the Standard group of the Toolbox and dropping it on the page in Design view or Source view. By default, a text box accepts one line of text. You can limit the number of characters the user can enter by opening the properties page (F4) and setting the MaxLength value.

Accepting passwords (somewhat) securely

When you set the TextMode property to Password, the text box hides the password from onlookers by substituting asterisks or bullets. In Figure 5-1, the second text box from the top shows the effect in the browser.

Capturing text with MultiLine mode

When you set the TextMode property to MultiLine, ASP.NET generates an HTML Textarea control. As shown in the bottom text box (refer to Figure 5-1), you set the number of visible lines with the value of the Rows property.

You can’t restrict the number of characters the user types into the TextBox control in MultiLine mode. See Chapter 19 for how to handle this.
Allowing creativity with rich text

An ASP.NET TextBox actively discourages rich text such as italic and bold. If you enter the following markup, ASP.NET complains about a “potentially dangerous” value.

\[ \text{I’m } \textit{entering} \text{ markup the } \textbf{hard} \text{ way.} \]

For details on dealing with the built-in protection, see Chapter 19.


Pushing for Choices with the RadioButton Control

ASP.NET RadioButton controls work as a team; however, only one player can be “on” at a time. Figure 5-2 shows three RadioButton controls acting as a group. All three share the same GroupName value. When a user clicks the Submit button, an event handler subroutine (refer to the “Bingo! And events” sidebar) executes and reports which radio button is checked.

![Figure 5-2: You can select only one radio button in a group at a time.](image)

Follow these steps to create a group of RadioButton controls and display which one a user has pushed:

1. From the Toolbox, add to the ASP.NET page three RadioButton controls, a Button control (Button1) and a Label control (lblText).
2. Set the RadioButton control’s ID values to radTo, radMtl, and radVcr; the Text properties to Toronto, Montreal, and Vancouver; and the GroupName properties to cities.

3. Double-click the button to create a handler for the Button control’s Click event and use the following code inside the Click event handler subroutine:

```vbnet
If radTo.Checked Then
    lblText.Text = "Choice: " & radTo.Text
ElseIf radMtl.Checked Then
    lblText.Text = "Choice: " & radMtl.Text
ElseIf radVcr.Checked Then
    lblText.Text = "Choice: " & radVcr.Text
Else
    lblText.Text = "No choice made."
End If
```

The code tests whether the Toronto radio button’s Checked property is True (that is, whether the button is pushed). If so, it assigns a text value to the Label and the work is done. If the first button’s Checked property is False, the logic continues to the ElseIf keyword (it drops through in geekspeak) and tests the Montreal button, and so on. If the code reaches the Else part without finding a button that’s pushed, it reports the failure to make a choice.

**Collecting RadioButtonList Controls**

The ASP.NET RadioButtonList control allows you to create many radio buttons with one control. In this section, you build a survey form, work with the Collection editor, and hook up an event handler.

**Creating the basic page interface**

The survey interface consists of a prompt, a set of radio buttons as choices, a button, and an area for a response. Follow these steps to create the basic interface.

1. In the ASP.NET page Design view, add a Label control with the ID lblPrompt and set the Text value to Rate Your Fear of the Borg.
2. From the Toolbox, drop a RadioButtonList control on the design surface and set its ID to rblBorg.
3. Add another Label with the ID lblResponse and a Button control.
You add questions to the survey’s user interface in the next section.

Bingo! And events

Think of a game of Bingo where players are filling their cards with markers. Suddenly, a hand shoots into the air and a player shouts, “Bingo!” That’s an event.

Consider the player with the filled card as an ASP.NET control that raises an event called Bingo. The game’s assistants are event handlers who intervene when someone claims to have a full card. The following pseudo-code (unusable code that represents a programming idea) shows how you might handle a Bingo event.

```vbnet
Protected Sub BingoPlayer1_Bingo(ByVal player As Object, ByVal e As System.BingoEventArgs)
    Dim blnIsValidBingo as boolean
    Dim walker as New Assistant()
    blnIsValidBingo = walker.Verify(e.Card)
End Sub
```

In ASP.NET, when someone clicks a button, the button doesn’t shout, “Bingo!” It raises a Click event. If no code is on the page to handle the event, nothing much happens. However, if a designated event handler for the mouse click is on the page, the handler subroutine goes into action. That action could be changing a label’s color from blue to red or sending the accumulated data to a database.

You add questions to the survey’s user interface in the next section.

Adding list items with a Collection editor

You can add items to a RadioButtonList control at design-time by using a designer called Collection editor. Collection editors mostly work alike, regardless of the collection type. Follow these steps to design options for a questionnaire:

1. Click the RadioButtonList control’s Smart Tag arrow, and from the menu, choose Edit Items.

   The ListItem Collection editor opens.

2. Click the Add button (on the lower-left side).

   As shown in Figure 5-3, ListItem appears in the Members area on the left. Notice the 0 preceding the ListItem. The first item in a .NET collection is numbered zero. See the “The Borg and .NET collections” sidebar for more.

3. In the properties area on the right, set the Text value to Plenty and the Value property to 3.
4. Add three more items to the collection and set their **Text** and **Value** properties as follows:

<table>
<thead>
<tr>
<th>Text</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Somewhat</td>
<td>2</td>
</tr>
<tr>
<td>Whatever</td>
<td>1</td>
</tr>
<tr>
<td>Zilch</td>
<td>0</td>
</tr>
</tbody>
</table>

5. Click OK to close the ListItem Collection editor.

As shown in Figure 5-4, the user interface elements are in place. In the next section, you add some logic and interactivity.

---

**Capturing the survey choice**

So far, the survey form is just a (Vulcan-like) interface with no logic. Follow these steps to capture the user’s choice and show that choice in the browser:
1. In Design view, double-click an empty part of the page to create an event handler for the Page object’s Load event.

The IDE automatically inserts the following event handler code (formatted differently here) into the page:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
End Sub
```

2. In the line above the End Sub keywords, insert the following code:

```
lblResponse.Text = rblBorg.SelectedValue
```

When you run the page and click the button, the click causes the page to submit its data (a postback). A Page Load event occurs (fires in geekspeak) just before ASP.NET completes construction of the page. The Load event handler code looks at the RadioButtonList (rblBorg) and extracts whatever is in its SelectedValue property. The code assigns the SelectedValue value as the Text property of the Label so the user can see the results.

### The Borg and .NET collections

Fans of the science fiction series *Star Trek* know all about the Borg, those gray technoinvaders who wander around muttering, “Resistance is futile.” A .NET collection resembles The Borg Collective in that items within a collection are similar but have distinguishing characteristics (such as different machine parts).

You deal with members of a collection as a set or group. Your code can examine each member one by one from first to last. In geekspeak, the action of flipping through the set is *iterating* through a collection. The *For Each* loop is frequently used to give collections an efficient once-over. Like you can with cyborgs, you can refer to members of a .NET collection by an index number that reflects their position within the collective, *er* collection. One notable thing about collections in .NET is that their numbering is *zero-based*. That means the index number of the first item is 0. The index number of the second item is 1. Imagine the chaos within the Borg Collective when you infuse it with the knowledge that Seven of Nine is actually a Six of Nine in .NET’s zero-based counting.

### Checking CheckBox and CheckBoxList Controls

The CheckBox and CheckBoxList controls permit multiple choices. Unlike radio buttons, you can switch a check box on or off without affecting any of the other check boxes on the page.
Creating an arbitrary number of check boxes

The CheckBoxList control (like the RadioButtonList) is well suited to database applications where the number of items varies. In this section, you hook up (bind in geekspeak) a CheckBoxList to data.

To create a data-driven CheckBoxList, follow these steps:

1. From the Toolbox, drop a CheckBoxList control, Button control, and Label control on a Web form.

2. In the Properties window for the CheckBoxList control, set the RepeatColumns value to 2 and set the RepeatDirection value to Horizontal.

   These settings display the data in a two-column table.

3. Double-click a blank area of the page to create a handler for the Page object’s Load event and insert the following code:

   ```vbnet
   If Not IsPostBack Then
   Dim arrlGames As New ArrayList
   arrlGames.Add("Scrabble")
   arrlGames.Add("Crosswords")
   arrlGames.Add("WonderWord")
   arrlGames.Add("Sudoku")
   arrlGames.Sort()
   CheckBoxList1.DataSource = arrlGames
   CheckBoxList1.DataBind()
   End If
   ```

   The preceding adds items to a list, sorts the list, and tells the CheckBox to use the list for its data. Notice that the whole routine is wrapped in an If...End If sequence that tests the IsPostBack property. You want to fill the data only when the page first loads, not on each postback. Otherwise, you get duplicate games.

   For a discussion of the logic used in the keyword Not, see Chapter 14.

4. Switch to Design view, and double-click the Button to create a handler for its Click event and add the following code in the line above the End Sub:

   ```vbnet
   Dim strSel As String = ""
   For Each chbx As ListItem In CheckBoxList1.Items
   If chbx.Selected Then
   strSel = strSel & chbx.Text & "<br />
   End If
   Next
   Label1.Text = strSel
   ```
The preceding uses a `For Each` loop to look through the collection of `TextBox` items and create a string of text.

Run the page, check some games, and click the button to see what’s selected.

**For Each and the collection**

The sidebar, “The Borg and .NET collections,” refers to the `For Each` loop that you see in action inside the `Button1_Click` routine. Here’s the line of code from Step 4 that begins the sequence:

```vbnet
For Each chbx As ListItem In CheckBoxList1.Items
```

It helps to parse the line starting at the far right to put the code into English. It says, “Here’s a collection of items. You know that each of these items is a `ListItem` type. Let the variable `chbx` (at least for now) represent the first `ListItem` in this collection. Now move to the next line of code.”

With `chbx` representing the first item within the collection, you can examine the item’s `Selected` property. If the `CheckBox` has been checked, the `Selected` property’s value is `True` and you therefore proceed inside the `If` statement to find the following line:

```vbnet
strSel = strSel & chbx.Text & "<br />"
```

Again, it helps to look to the right side of the code to describe what’s happening. Here, you peer into the value of the `Text` property for the `CheckBox` (for example, “Crosswords”). You take that text, attach an HTML carriage return, and add this onto whatever is in the `strSel` variable. (On the first loop, nothing is in `strSel`.)

After exiting the `End If` statement, you run into the keyword `Next`. `Next` says, “Okay folks, we’re done with that member of the collection, let’s do the same thing with the next one.” The sequence repeats until the `For Each...Next` combination announces, “It’s quittin’ time ‘cause we’re fresh outta check boxes.”

**Using the DropDownList Control**

The ASP.NET `DropDownList` control displays a large number of items in a very little space because it drops down to display its list when the user clicks the arrow. (Sometimes, it *rises upward* to display the items.)
At design-time, you can add static items to the DropDownList control by using the ListItem collection editor. At runtime, you can fill a DropDownList control with almost any data as long as you can get it into a simple list. To put color names in a DropDownList control, follow these steps:

1. **From the Toolbox, add a DropDownList, Label, and Panel control to an ASP.NET page.**

2. **Select the DropDownList control and set its AutoPostBack property to True.**

   AutoPostBack causes a page to submit its data to the Web server (and cause a postback) when the user merely selects a different item. No Submit button is required.

3. **Double-click the DropDownList control to create its default event handler and use the following code inside the SelectedIndexChanged subroutine:**

   ```vbnet
   Dim strClr As String
   strClr = DropDownList1.SelectedValue
   Dim objColor As System.Drawing.Color
   objColor = System.Drawing.ColorTranslator.FromHtml(strClr)
   Panel1.BackColor = objColor
   Label1.Text = strClr
   ```

4. **Return to Design view and double-click a blank area of the surface to create an event handler for the Page object's Load event and then add the following code above the final line of the Page_Load routine:**

   ```vbnet
   If Not IsPostBack Then
   Dim enClr As System.Drawing.KnownColor
   clrs.AddRange(System.Enum.GetValues(enClr.GetType()))
   DropDownList1.DataSource = clrs
   DropDownList1.DataBind()
   Panel1.Height = Unit.Pixel(200)
   Panel1.Width = Unit.Pixel(300)
   End If
   ```

When you browse to the page, the drop-down list fills with dozens of color names. Make a selection. The name and its color swatch appear on the screen. Walk through the code to see how it works.
Understanding namespaces

The .NET system (on which ASP.NET is based) is thousands of useful chunks of code organized into categories called namespaces. For example, in the code for the Page Load event, you see this line:

```
Dim enClr As System.Drawing.KnownColor
```

The namespace used in the preceding code is System.Drawing. The Web server’s hard drive has a system.drawing.dll file, which is where the System.Drawing code resides. In geekspeak, system.drawing.dll is known as an assembly. Within this namespace is a list of system-defined colors, such as YellowGreen.

Retrieving a list of colors

When the page loads the first time, you declare the variable `enClr` as a KnownColor type. Next, you create a generic list that works easily with ASP.NET controls. You stuff the color values into the list. Finally, you instruct the DropDownList control to get its data from the list. When you fill the DropDownList with data, the control automatically retains the values (persists in geekspeak). Therefore, you fill the data on the initial page load, not on each postback.

Displaying the color name and showing the color

When the user changes the DropDownList, the SelectedIndexChanged event fires and your event handler goes into action. In this routine, you capture the name of the selected color in the variable `strColor`. Next, you declare the variable `objColor` as a System.Drawing.Color type so it can hold that type of content.

Converting a color name, such as YellowGreen into a Color type is a little tricky. Inside the System.Drawing namespace is a useful chunk of code (a class in geekspeak) called ColorTranslator. One of the capabilities of ColorTranslator (the FromHtml() method) takes a name or value that’s in an HTML format (such as #ff00aa or White) and converts it to a .NET Color.

After you convert the ordinary color name into something that the Panel control understands, you tell the Panel control to use that for its background color (BackColor). As for the Label control, you already have the name of the color, so you instruct the Label to display the name as its Text property.
Getting Multiple Choices from a ListBox

The ListBox control shows several items at a time inside a box. You set the number of visible items by using the Rows property. Users can select more than one item by holding down the Ctrl key while clicking the items. This example allows users to choose and display font names. Follow these steps to create the font list box:

1. **From the Toolbox, add a ListBox, Button, and Label control to the Web page.**
2. **Select the ListBox and, in its Properties window (F4), set the SelectionMode property to Multiple.**
3. **Double-click an empty area of Design view to create a handler for the PageLoad event and add the following LINQ query to fill the ListBox with font names from a built-in .NET collection:**
   ```vbnet
   If Not IsPostBack Then
   Dim q = From f In System.Drawing.FontFamily.Families 
   Select f.Name 
   ListBox1.DataSource = q 
   ListBox1.DataBind()
   End If
   ```
   For details on LINQ syntax, see Chapter 7 and this book’s cheat sheet.
4. **Add the following Imports directive to the top of the page in Source view:**
   ```vbscript
   <%= Import Namespace="System.Linq" %>
   ```
5. **Return to Design view and double-click the Button control to create a Click event handler and add the following code:**
   ```vbscript
   Dim strItems As String = ""
   For Each itm In ListBox1.Items
   If itm.Selected Then
   strItems = strItems & itm.Text & "<br />"
   End If
   Next
   Label1.Text = strItems
   ```

When you browse the page, the ListBox displays the server’s fonts. Select a few fonts and click the button to show their names.
Understanding ASP.NET Forms

In ASP.NET, server controls, such as the TextBox and DropDownList, must reside within a server-side form. In Design view, the development environment knows this rule and inserts controls in the right place.

To understand forms, it helps to analyze the behind-the-scenes markup. Listing 5-1 shows the code that appears in Source view when you add a single page called myform.aspx to your project.

Listing 5-1: The myform.aspx Source Code

```html
<%@ Page Language="VB" %>
<DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
</div>
</form>
</body>
</html>
```

1 The line `<%@ Page Language="VB" %>` is a Page directive. It provides important information to ASP.NET while it compiles and assembles the page on the server. In this case, the Language attribute’s value is `VB`, meaning ASP.NET should expect Visual Basic code. This and other directives aren’t sent as HTML to the browser.

3-6 The markup starting with `<!DOCTYPE` and ending with `dtd">` is sent to the browser as is. It describes the HTML standard to which this page conforms.

7-9 The markup `<script runat="server"></script>` includes the important runat attribute with the value `server`. Computer code within these tags is processed on the Web server. The browser sees the results of the process.
The \texttt{<html>} tag goes directly to the browser without processing because \texttt{runat="server"} isn't present.

The \texttt{<head>} tag includes \texttt{runat="server"}, which means that the Web server's process knows about the tag's contents.

After the familiar HTML \texttt{<body>} tag, comes the all-important \texttt{<form id="form1" runat="server">} markup.

The rest of the markup is standard HTML and mainly closing tags.

Even though this page does absolutely nothing, it's instructive to run it and look at what the browser sees. Follow these steps to run the page and view the HTML:

1. In Visual Web Developer, add a Web form called \texttt{myform.aspx} to your application (File\ процедура New File\ процедура Web Form (myform.aspx)\ процедура Add).
2. Browse to the page and view the source code. (In IE 7, choose View\ процедура Source. If Windows Vista asks for permission, give it.)

Some strange things happen to the code during the server processing:

- The page directive (@ Page) is missing. That's a server-side instruction so the browser doesn't see it.
- The \texttt{<script runat="server">} markup is gone. It's another server-side instruction.
- The \texttt{<form>} tag survived but has \texttt{method} and \texttt{action} attributes that weren't there before. The server has generated these and pointed the \texttt{action} attribute at the \texttt{myform.aspx} filename.
- As the following code shows, there's now a hidden \texttt{<input>} tag called \texttt{__VIEWSTATE} with a long encoded value that wasn't in the ASP.NET source page:

  \begin{verbatim}
  <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMTUxMzcyMjQyN2RkCzHRdRR1uHRoA8uH8qQCo0hGTAI=" />
  \end{verbatim}

\texttt{Viewstate} is ASP.NET's sleight of hand. It encodes information about the current state of the page and its controls. The next time the server sees the page, it reads the encoded information; and from that, figures out what changed, what was clicked, and what was selected.

This drives home the fact that server-side code and client-side code are somewhat the same but live in different worlds.
Part II

Immersing Yourself in Data

The 5th Wave

By Rich Tennant

"I'll be with you as soon as I execute a few more commands."
In this part . . .

In this data-rich part, prepare for something old, something new, something borrowed, and something blue. The *old* is in Chapter 6, where you use the SqlDataSource control to manipulate the Northwind database. The *new* is the exciting introduction of LINQ, which I cover in Chapters 7 and 8. The marriage of Visual Basic and a dedicated query language is worth celebrating. If you find the SQL language difficult and error-prone, vow to embrace LINQ syntax until death you do part. By the way, to get your head around the new LINQ query syntax, tear out the handy cheat sheet inside the front cover and tape it to the bottom of your monitor.

The *borrowed* appears in Chapter 9, where you display an RSS data feed borrowed from another site. Finally, *blue* enters the picture in the Web service sample that calculates red, blue, and green values from a color name.
Chapter 6

Fetching and Presenting Data with SqlDataSource

In This Chapter

- Using SQL Server Express as a data source
- Building connection strings
- Using the SqlDataSource control
- Passing parameters from controls and other sources
- Creating a master/detail page

In Chapter 3, I show you how to create a database and, using the power of the Visual Web Developer environment, generate a data-driven Web page. This chapter still emphasizes letting the tools do the work, but the approach is different. The goal is to understand what the wizards are doing so you can use them more effectively in a variety of situations.

Connecting to SQL Server Express

Data connections are easy on days when your biorhythms are running high. Fortunately, after you get a connection working, you can set it and forget it.

Checking whether SQLExpress is running

This section assumes that you installed SQL Server 2005 Express (SQLExpress) on your workstation. Installation is covered in Chapter 2.

Before you try connecting to SQL Server Express it helps to know whether the SQL software is running. Follow these steps to use a command line utility to check your system for a running instance of SQL Express:
1. **Open a command prompt:**
   - If you’re using Windows XP, choose Start Run; enter cmd and press Enter.
   - If you’re using Windows Vista, choose Start, enter cmd in the search box, and press Ctrl+Shift+Enter.

2. **At the command prompt, type the following command:**
   
   sqlcmd -S(local)\SQLExpress

3. **Press Enter.**
   
   If SQLExpress is running, the program responds with a prompt that looks like
   
   \[1>\]
   
   If SQLExpress isn’t running, the program reports a connection error and quits.

4. **Type exit and press Enter to exit the sqlcmd utility and then type exit and press Enter again to close the command line utility.**

   If SQLExpress shows no sign of life, rerun the VWD installer to repair it. If you know SQLExpress is installed but hasn’t started, try the following command from the command prompt:

   net start "SQL Server (SQLEXPRESS)"

   Microsoft’s Web site has several articles to help with starting SQL Server. Try a search for mssql\sqlExpress faq as a, er, starting point.

---

**Finding a copy of the Northwind database**

You can follow along in this chapter using almost any SQL Server 2005 database, including one that you build yourself. However, it’s much easier to compare your results if you use Microsoft’s ever-popular Northwind database. Browse to http://www.microsoft.com/downloads and search for Northwind and pubs Sample Databases for SQL Server 2000. After double-clicking the downloaded file to run its installer, you should find the northwnd.mdf file in C:\SQL Server 2000 Sample Databases.

---

**Adding the Northwind database to your application**

Visual Web Developer reserves a special folder called App_Data for storing SQL Express database files. To add the Northwind database to your Web application, do the following:
1. Add an App_Data folder to the project if it doesn’t exist (Website › Add ASP.NET Folder › App_Data).

2. In Solution Explorer, click the App_Data folder and choose Website › Add Existing Item.

3. Navigate to the Northwind database file (for example, C:\SQL Server 2000 Sample Databases \northwnd.mdf) and click Add.

Connecting to the database

Your Web pages — or more accurately, your data controls — must be able to find the database engine along with the data file. Follow these steps to check for and add a data connection:

1. In Visual Web Developer, open Database/Server Explorer (View › Database/Server Explorer).

2. Expand the Data Connections node and look for a node called northwnd.mdf.

   A red X on the database icon indicates the connection might not be open or working. It might just be resting.

3. If the northwnd.mdf node exists, expand the node (it can take a few seconds to respond) and then expand the Tables node, as shown in Figure 6-1, to confirm that the connection is working.

   If it’s working, you’re connected, and you can skip the remaining steps.

4. If there’s no working connection, right-click the Data Connections node, and choose Add Connection from the context menu.

   The Add Connection dialog box appears, as shown in Figure 6-2.
5. Next to the Data Source box, click the Change button.

The Change Data Source dialog box appears, as shown in Figure 6-3.

6. Select Microsoft SQL Server Database File and then click OK.

7. In the Add Connection dialog box, next to the Database File Name box, click the Browse button and navigate to the copy of the Northwind database that’s in your App_Data folder.

You can determine the path by selecting northwnd.mdf in Solution Explorer and looking at the Full Path property in its Properties window.

8. To make sure that you have a good connection, click Test Connection in the lower left of the Add Connection dialog box, and then click OK.

At this point, you have a working data connection and you’re ready to use the SqlDataSource control.
Using the SqlDataSource Control

The SqlDataSource control is a user-friendly way of working with Microsoft’s ADO.NET data handling technology. It does much of the grunt work for you, such as opening a connection to the database, executing a SQL statement, fetching the data, and closing the data connection.

Adding and configuring a SqlDataSource control

You need a working SqlDataSource control so that other controls, such as GridView, FormView, and ListView, can use it for their data needs. To add a SqlDataSource control to your page, follow these steps:

1. Add a single file ASP.NET Web Form (don’t use the Place Code in Separate File option) called starter.aspx to your project.

2. Drag a SqlDataSource control from the Data category of the Toolbox and drop it on the page.

3. Click the Smart Tag button and select Configure Data Source.

   The Configure Data Source Wizard appears.

4. From the drop-down list, choose the northwnd.mdf data connection and then click Next.

5. Accept (that is, leave checked) the offer to save the connection string in the Save the Connection String screen, and then click Next.

6. In the Configure the Select Statement dialog box, choose the Customers table from the drop-down list.

   The columns (fields) in the Customers table appear in the Columns box.

7. In the Columns box, select the check box for the asterisk.

   This indicates that you want to select all the columns for the query.

8. Click the Advanced button.

   The Advanced SQL Generation Options dialog box appears, as shown in Figure 6-4.

9. Select the Generate INSERT, UPDATE, and DELETE Statements check box (do they need to shout?), and then click OK.
10. Back in the Configure the Select Statement screen, click Next.

11. In the Test Query screen, click Test Query, and then click Finish.

You now have a SqlDataSource configured to fetch and update data. At this point, nothing on the starter.aspx page allows users to see or interact with the data control. You fix that later in this chapter in the section “Consuming Data with the DetailsView Control.”

In the preceding steps, the SqlDataSource Wizard inserted declarative markup — the HTML- or XML-like code into the .aspx file. To view the generated code, select the SqlDataSource control in Design view and then switch to Source or Split view.

The ConnectionString attribute

The SqlDataSource control needs to know where to get its data. To follow the trail, open the page in Source view and locate the following markup:

```
ConnectionString='<%$ ConnectionStrings:ConnectionString %>'
```

In plain English, the markup says, “At runtime, go look in the web.config file for a section called ConnectionStrings. After you find it, look for an attribute called ConnectionString and bring back its value. Jam that value between the quotation marks you see here. When I need to know where to get my data, I’ll refer to that source.”

The first ConnectionString is the attribute declaration. After the equal sign (=) comes the value in quotations marks. The stuff inside the quotation marks isn’t the connection string; it’s a just placeholder for a future connection string. Here’s the deal: The tags `<%...%>` tell ASP.NET to wait until runtime to evaluate the content. The dollar sign ($) indicates that the part to be evaluated is found in the web.config file. The ConnectionStrings: portion (note the final “s”) describes the section of the web.config, and the final ConnectionString points to the attribute where the connection string is stored.
The connectionString section of the web.config file

The following snippet from the web.config file shows all the required parts of a connection string. Granted, it looks messy, but it makes sense to the SqlDataSource control (after all, its wizard wrote this code), and that’s what matters.

```
<connectionStrings>
  <add name="ConnectionString" connectionString="Data Source=\SQLEXPRESS;AttachDbFilename=|DataDirectory|\northwnd.mdf;Integrated Security=True;Persist Security Info=True;
  Connect Timeout=30;User Instance=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

At runtime, ASP.NET replaces |DataDirectory| with the physical path to the special App_Data folder.

The Command attributes in the markup

In starter.aspx, you probably recognize the SQL in the DeleteCommand, InsertCommand, SelectCommand, and UpdateCommand attributes embedded in the SqlDataSource control’s markup. The DeleteCommand’s value looks like this:

```
DELETE FROM [Customers] WHERE [CustomerID] = @CustomerID
```

The @CustomerID part is a fill-in parameter. Parameters are handy when you don’t know what the value is going to be until the last instant. For example, you have to wait until a user clicks a Delete button to know which CustomerID they want to remove.

Defining parameters

The declarative markup defines parameters for each type of command. Here’s the definition for the @CustomerID parameter used by the DeleteCommand:

```
<deleteparameters>
  <asp:parameter Name="CustomerID" Type="String" />
</deleteparameters>
```

Each asp:parameter provides you its name (CustomerID) and type (String). In some cases, you can include the DefaultValue attribute to indicate the value to use when the control doesn’t supply one.
Consuming Data with the DetailsView Control

So far, in this chapter, you’ve installed the database, connected the SqlDataSource to the database, and configured the SqlDataSource control so it can fetch, update, and insert records. In this section, you connect an ASP.NET DetailsView control to the SqlDataSource control to create a user interface.

The ASP.NET DetailsView control displays details one record at a time from a data source. It has built-in support for updating, inserting, deleting, and paging through data. Follow these instructions to add a DetailsView control to your page and hook it up to the data:

1. **Open the starter.aspx page (created previously) in Design view.**
2. **Drag a DetailsView control from the Data category of the Toolbox, and drop it on the page.**

   Figure 6-5 shows the DetailsView control with its initial Tasks menu.

3. **From the drop-down list, choose the SqlDataSource control that you configured in the earlier section, “Adding and configuring a SqlDataSource control.”**

   As shown in Figure 6-6, the DetailsView scans the contents of the SqlDataSource control and includes the fields in its list.

4. **Select the options to enable paging, inserting, editing, and deleting.**

   When you run the page, the DetailsView control shows the first customer with navigation links to the remaining records. Figure 6-7 shows the Edit, Delete, and New buttons near the bottom of the page.
The DetailsView control is clumsy when navigating through many records. By displaying only one record at a time, you spend a long time paging to the last customer in the database.

### Database orders: Don’t delete my customers!

If you try to delete a customer in the Northwind database, you’ll likely get this error:

```
The DELETE statement conflicted with the REFERENCE constraint "FK_Orders_Customers".
```

SQL Server complains because the database contains a list of orders in the Orders table that belongs to the customer you’re trying to delete. The reference to the customer is a foreign key (FK for short). If you delete the customer first, the database ends up with a messy pile of orphaned orders and order details. You work around a deletion constraint in Chapter 8.
You can improve the performance of the DetailsView control’s paging by opening its Properties window and setting the EnablePagingCallbacks property to True.

Using Parameters in Queries

When you set up the SqlDataSource control’s SelectCommand (see the earlier section, “Adding and configuring a SqlDataSource control”) you told it to fetch all the customers. You probably don’t want to deal with that much data at one time because the more records, the more paging. The preferable method is to tell the SqlDataSource control to limit the scope of the data. You filter the data by passing one or more parameters. As you see in this section, you can get parameter values from many sources.

Getting a parameter value from a TextBox control

Say you want to filter the database query on the starter.aspx page to show only customers from a given country. You only know the country name at runtime so you can’t hardcode the name. The easiest way is to let the user type the country name and use that value as part of the query. Follow these steps to pass a parameter value from an ASP.NET TextBox control to the SqlDataSource control:

1. From the Toolbox (in the Standard category), add an ASP.NET TextBox to the page (above the DetailsView control) and set its ID property to txtCountry.
2. Add an ASP.NET Button control to the page.
3. Open the SqlDataSource control’s Tasks menu and choose Configure Data Source.
4. In the Configure Data Source Wizard, click the Next button to step to the Configure the Select Statement screen and click the WHERE button.
   The Add WHERE Clause dialog box appears.
5. From the Column drop-down list, choose the Country field.
   This sets Country as the database field on which you want to filter.
6. From the Operator drop-down list, select LIKE.
7. From the Source drop-down list, select Control.
8. In the Parameter Properties area, from the Control ID drop-down list, choose `txtCountry`.

Figure 6-8 shows how the Add WHERE Clause screen looks based on the choices made so far.

![Figure 6-8](image)

9. Click Add.

The wizard displays the SQL expression.

10. Click OK and step through the remaining dialog boxes to finish.

When you run the page, type **Canada** in the text box and click the button. The query returns the records of three companies from Canada.

This query used the **LIKE** operator, which can be convenient and misleading at the same time. It’s convenient because you can enter just part of a country name (for example **Fr** for France) and get results. Misleading, because it matches characters anywhere in the country name, and this can lead to odd results. For example, if you type just **t**, you’ll get back Argentina and all the other countries that have a **t** in their names.

All user input is evil until proven otherwise. Chapter 19 shows you how to secure the text box from malicious or troublesome input.

You can make the country filter user friendly by letting the user pick from a list of the countries represented in the database. As they say on television, it’s coming right up.
Returning the country names with no repeats

The goal in this section is to display all the country names in a drop-down list. Then the DetailsView should only show companies from the selected country. The first step is to fill the drop-down list with countries from the database. Follow these steps to configure another SqlDataSource control:

1. **From the Toolbox, add a** SqlDataSource **called SqlDCountries to the** starter.aspx **page.**

2. **Using the Smart Tag button, choose Configure Data Source.**

3. **In the Choose Your Data Connection screen, select the data connection that you used for the preceding SqlDataSource (probably ConnectionString) and then click Next.**

4. **In the Configure the Select Statement dialog box, from the Name drop-down list, choose the Customers table, as shown in Figure 6-9.**

![Figure 6-9: The unique rows option eliminates duplicate country names.](image)

5. **In the Columns box, select the Country check box (refer to Figure 6-9).**

6. **Check the Return Only Unique Rows option (middle right of the screen).**

7. **Click ORDER BY and in the Add ORDER BY Clause dialog box, select Country from the drop-down list and click OK.**

8. **Click Next and complete the remaining wizard steps.**
At runtime, the SqlDSCountries control looks through the Customers table for the names of countries and brings back a sorted list with no repeats.

**Filling a drop-down list with data from a SqlDataSource**

After you have a SqlDataSource that can get the country names, you can display the names to the user in a drop-down list. Follow these steps to add and configure the control:

1. From the Toolbox, drag and drop an ASP.NET DropDownList control on the page and set its ID property to ddlCountries.
2. In the Properties window, set the DataSourceID property to SqlDSCountries.
3. Set the DataTextField to Country.
4. Set the DataValueField to Country and then click OK.

The preceding steps tell the drop-down list to get the visible text and the underlying value from the same place, the Country field.

**Changing the parameter source**

The original SqlDataSource control that fetches the customer data is now looking in the wrong place for its parameter. Follow these steps to reorient the parameter source to get the parameter from the drop-down list:

1. Select the original SqlDataSource control (most likely with the SqlDataSource1 ID).
2. In the SqlDataSource Properties window (F4), select the SelectQuery property and, in the right-hand column, click the ellipsis (...) button.
   The Command and Parameter Editor opens, as shown in Figure 6-10.
3. In the SELECT Command box, replace the existing command with the following:
   ```sql
   SELECT * FROM [Customers] WHERE ([Country] = @Country)
   ```
4. From the Parameter Source drop-down list, choose Control.
5. From the ControlID drop-down list, choose ddlCountries.
6. Click OK.
7. Back on the ASP.NET page, select the TextBox control, txtCountry and delete it by pressing Delete.

That’s it! You’ve told the SqlDataSource control to get its Country parameter from the drop-down list. The country names appear as choices at runtime. Choose a country, and click the button — the DetailsView control displays the first match with navigation links to step through the remaining records.

**Obtaining a parameter from a Session variable**

A Session variable is a handy way of storing a value on the Web server while the user is browsing the site. Instead of prompting a user for their address or phone number repeatedly, your page can collect the data one time, store it as a Session variable, and ask the Web server to provide it when required.

The SqlDataSource control can read a value from a Session variable and use it as a parameter value. In this section, you store a value in a Session variable in one page and fetch the value as a parameter in a separate page. You reuse the DetailsView page (starter.aspx) from the previous section by reconfiguring the SqlDataSource control's Select parameter.
Session variables die when the user closes the browser. Even if the user leaves the browser open, the Session expires 20 minutes (the default) after the last page visit or refresh at the Web site.

To create a page that sets a Session variable, follow these steps:

1. Add a new ASP.NET page called Session.aspx to your project.
2. From the Toolbox, add TextBox, Button, and HyperLink controls to the page.
3. Select the HyperLink control and in its Properties window (F4), set the NavigateUrl property to starter.aspx, the page you used previously to display the DetailsView.
4. In Design view, double-click the Button control to create an event handler for the Click event.
   The IDE switches to Source view. (For more on event handlers, see Chapter 5.)
5. In the event handler for the Click event, in the line above the End Sub statement, type the following line of code:

   ```csharp
   Session("strCountry") = TextBox1.Text
   ```

   The preceding code fetches the value typed into TextBox1 and stores it in a Session variable on the Web server. Now that you have a way of setting the Session variable, you need to configure starter.aspx to read and use the stored value.

   To configure the SqlDataSource to use a Session variable, do the following:

   1. Open starter.aspx (the page from previous examples that includes the DetailsView control).
2. Remove the DropDownList control (its ID is ddlCountries) by selecting the control and pressing the Delete key.
3. Select the SqlDataSource control. (The ID is likely SqlDataSource1. Don’t use the one called SqlDSCountries.)
4. Open the SqlDataSource control’s Properties window (F4), select the SelectQuery property and, in the right-hand column, click the ellipsis button.
5. In the Command and Parameter Editor, from the Parameter Source drop-down list, choose Session. See Figure 6-11.
6. In the Session Field box, type strCountry, as shown in Figure 6-11.
7. Click OK.
In this example, the session.aspx page sets the Session variable and provides a way to navigate to the DetailsView page (starter.aspx) where the SqlDataSource controls picks up the Session value and uses it as its parameter. Here’s how to test the pages:

1. **Browse to** session.aspx.
2. **In the text box, type France and click the button.**
   Nothing much appears to happen, but behind the scenes, the button’s event handler set the Session variable to France.
3. **Click the hyperlink to navigate to** starter.aspx, **which contains the DetailsView control.**
   The page opens with the DetailsView control showing the first customer from France and navigation links to the remaining French customers.

**Session variables are handy but too many can gobble up the Web server’s memory and cause the server to misbehave.**

**Passing a parameter on a query string**

You’ve certainly seen thousands of query string parameters even if you didn’t know what they’re called. The following is a URL from a famous ASP.NET Web site. The query string starts at the question mark (?) and continues to the end:
This query string includes a name/value pair. The name in this case is `catId` and the value is `c02`. When the Web server sees a query string, it passes the information to ASP.NET for further processing.

The `SqlDataSource` control can pick out the value and use it as part of a SQL query. Follow these steps to create a query string that passes a value:

1. Open `starter.aspx`, the ASP.NET page that includes the DetailsView control.
2. From the Toolbox, add an ASP.NET HyperLink control to the page.
3. In the HyperLink control's Properties window, insert the following value for the NavigateUrl property:
   ~/starter.aspx?country=UK

When you click the hyperlink, the browser requests the current page (`starter.aspx`) and at the same time passes the name/value pair `country/UK`. Follow these steps to tell the `SqlDataSource` control to use the query string value when getting data:

1. Select the `SqlDataSource` control (the ID is likely `SqlDataSource1` and not `SqlDSCountries`).
2. In the `SqlDataSource` control's Properties window (F4), select the SelectQuery property, and click the ellipsis button in the right-hand column.
3. In the Command and Parameter Editor, choose QueryString from the Parameter Source drop-down list.
4. In the QueryStringField box, type `country` and then click OK.

When you run the page and click the hyperlink, the `SqlDataSource` reads the value (UK) from the query string and uses it as part of the SQL query.

Creating a Master/Detail Page

Thanks to VWD's wizards and designers, you can assemble a master/detail page faster than a Nipissing Township deerfly finds a swimmer's wet head. (Visit http://nipissingtownship.com for more on our rural community.)

In your master/detail page (shown in Figure 6-12), you select a customer from the master `DataGrid` on the left, and view or edit the details in a Details View control on the right. Geeks, who have obviously never held a power drill, call this scenario *drilling down* to the detail.
Designing the page layout

The page design lists customers on the left and the details on the right. To create a basic two-column page layout, follow these steps:

1. In Solution Explorer, add an ASP.NET page called mdtlpg.aspx to your project (File ➪ New File ➪ Web Form).

2. In Source view, just before the closing </head> tag, add the following style sheet markup:

```html
<style type="text/css">
  .leftcontent
  {
    float: left;
    width: 41%;
    padding-right:20px;
  }
</style>
```

3. In Design view, from the Toolbox, drag and drop a Panel control onto the page and set its CssClass property to leftcontent.

4. Drag and drop another Panel control onto the page to the right of the existing Panel.

5. Remove the default Height and Width properties and values from the Panel controls.

6. From the Toolbox, drag a GridView control and drop it inside the first Panel control (Panel1).

7. Drag a DetailsView control and drop it inside the second Panel control (Panel2).
Figure 6-13 shows the page designer with the Panel controls, GridView, and DetailsView. Don’t worry that the alignment isn’t correct. The designer rendering isn’t always accurate. The next step is to add the data controls.

**Fetching data for the master**

Getting the data for the GridView control requires a read-only configuration of a SqlDataSource control. This time, you only need to display the customer’s name and ID so that’s all the data you request. Follow these steps to configure the data source for the master portion of the page:

1. Be sure that Visual Web Developer has a working connection to the Northwind database.

   For help, refer to the earlier section, “Connecting to the database.”

2. Add a SqlDataSource control to the bottom of the ASP.NET page and set its ID property to SqlDataMaster.

3. Using the Smart Tag button, start the Configure Data Source Wizard.

   For help, refer to the earlier section, “Adding and configuring a SqlDataSource control.

4. Select the connection string (probably named ConnectionString) to the Northwind database that you configured previously.

5. In the Configure the Select Statement dialog box, specify the Customers table and check only the CustomerID and CompanyName columns.

6. Step through the remaining dialog boxes to complete the wizard.

The next task is to get data for the DetailsView by using a parameterized query.
Fetching data for the details

In this application, the DetailsView shows only details for the row that the user has selected in the GridView. That means the SqlDataSource needs a parameter to know which record to fetch. Follow these steps to add and configure a parameterized SqlDataSource to the page:

1. **Add a second SqlDataSource control to the page and set its ID property to SqlDataDetails.**
2. **Using the Configure Data Source Wizard, configure the SqlDataDetails to use the same data connection as the GridView control (probably named ConnectionString).**
3. **Configure SqlDataDetails to use the Customers table and fetch all (*) rows.**
4. Click the Advanced button and select the Generate INSERT, UPDATE, and DELETE Statements check box and then click OK.
5. Click the WHERE button to open the Add WHERE Clause screen.
6. Using Figure 6-14 as the model, create a parameterized query on the CustomerID column, using the equal (=) operator with the source as a control with the ID GridView1. (Don’t forget to click Add to save the expression.)
7. Click Next and Finish to exit the Configure Data Source Wizard.

Figure 6-14: Adding a WHERE clause and a parameter for the CustomerID column.
Configuring the GridView and DetailsView controls

The final task in the creation of the master/detail page is to point the ASP.NET controls to their respective data sources and set a few options. Follow these steps to configure the controls:

1. **Select the GridView control and using the Smart Tag, select SqlDataMaster as its data source.**

2. **On the GridView control’s Tasks menu, enable the paging, sorting and selection options, as shown in Figure 6-15.**

3. **Select the DetailsView control and set its data source to SqlDataDetails.**

4. **On the DetailsView control’s Tasks menu, enable the inserting and editing options, as shown in Figure 6-16.**
This master/detail page has the essential elements: When you click Select on the GridView, the company’s details appear in the DetailsView. You can click the Edit link to edit the data, and then click Update to save it.

An obvious enhancement is the addition of some AJAX to reduce annoying page refreshes. For more on the ScriptManager and UpdatePanel controls, refer to Chapter 4.
Chapter 7

LINQ as a Data Language

In This Chapter
- Using From, Where, and Select clauses
- Filtering, grouping, and narrowing scope
- Aggregating for minimum and maximum impact
- Creating and querying XML with LINQ
- Using object initializers

LINQ — Language Integrated Query — is a new way of dealing with data. In Visual Basic, LINQ creates a standard syntax and associated keywords that work the same way whether data comes from SQL Server, XML files, or in-memory objects. In LINQ, you treat data as objects by using their collections, properties, and methods. Instead of writing the traditional SQL Server syntax, you work with LINQ clauses, such as From, Select, and Where. Behind the scenes, LINQ converts those keywords and values into statements that SQL Server understands.

Those of us who struggle with trial-and-error SQL statements are likely to jump on the LINQ bandwagon because with LINQ, it’s harder to create errors. For example, the Visual Web Developer environment paints bad LINQ syntax with squiggly lines, and IntelliSense pops up LINQ keywords that make sense in the context.

Setting Up the LINQ Examples

This chapter walks you through LINQ examples, using arrays, collections, XML, and relational data. To avoid frustration with the LINQ to SQL examples, replicate the environment. This section helps you get started on the right foot.
Creating the **DataContext object**

Some of the examples in this chapter require the Northwind database and the DataContext object that supports LINQ queries. Follow these steps to generate the DataContext object for the Northwind database:

1. **Add the sample Northwind database and a working data connection to your project** (see Chapter 6).

2. **Add a LINQ to SQL Classes file called NWDataClasses.dbml to your project** (File ➤ New File ➤ LINQ to SQL Classes ➤ Add).
   
   If you’re prompted to put the file in a special App_Code folder, click Yes. VWD creates the folder.
   
   The split-page object relational designer appears.

3. **From Database Explorer (Server Explorer in non-Express versions), expand the Tables node.**

4. **As shown in Figure 7-1, drag the Categories, Order Details, Products, and Suppliers table names from Database (or Server) Explorer and drop them on the left-hand (larger) design surface of NWDataClasses.dbml.**

   This generates the DataContext code in NWDataClasses.designer.vb.

---

**Figure 7-1:**
Creating the DataContext code.

---

**Creating ASP.NET pages for the examples**

The emphasis in this chapter is on LINQ syntax and keywords rather than on creating complete and functional Web pages. Of course, you’ll want to try the
code in your own pages to analyze how the queries work. Follow these steps each time to create ASP.NET pages that use the results:

1. **Add an ASP.NET page to your project.**
2. **In Source view, add the following Import directive to the top of the page:**
   ```xml
   <%@ Import Namespace="System.Linq" %>
   ```
3. **In Design view, drop the type of control that the example uses for display.**
   Usually it's a GridView, BulletedList, CheckBoxList, ListBox, or Label.
   You can determine the type of control the sample uses by examining the snippet's DataSource property. For example, this one uses the GridView control:
   ```vbnet
   GridView1.DataSource = q
   ```
4. **In Design view, double-click a blank area of the page to create a default handler for the Page Load event.**
5. **In Source view, enter the sample code inside the Page_Load() subroutine.**
   You're ready to hook up with LINQ.

**LINQing with From, Where, and Select**

LINQ adds a bunch of keywords to Visual Basic. If you’ve worked with SQL statements, you’ll recognize many of the keywords, especially From, Select, and Where. This section shows how these popular clauses work, including samples of their syntax.

In a LINQ query, you declare a variable to hold the return value (for example, Dim q), but you don’t have to declare what type of variable it is (such as String or Boolean). LINQ uses **anonymous types** because it often deals with data that contains multiple types that aren’t compatible. When LINQ can’t tell you what type a query returns, LINQ just makes up a type and passes it off as anonymous.

In Visual Basic, a LINQ expression must appear as one statement; thus, one line can become very long. To break the line and improve readability, Visual Basic supports a space followed by an underscore character (_ _) as you see in the examples.
Targeting the source in a From...In clause

A From clause points to the source collection (data or objects) and declares an iteration variable. You use an iteration variable as a temporary representative for one item in a collection of items. In the following example, tz is the iteration variable, and System.TimeZoneInfo.GetSystemTimeZones returns the collection:

```csharp
Dim q = From tz In System.TimeZoneInfo.GetSystemTimeZones
```

The preceding statement goes In the collection of time zones and retrieves all the zones. The compiler infers that tz is a System.TimeZoneInfo type because that's what the GetSystemTimeZones() method returns. If you want to be more precise about the type, you can include the As keyword, followed by the type name:

```csharp
Dim q = From tz As TimeZoneInfo In System.TimeZoneInfo.GetSystemTimeZones()
```

To get at the content of the objects, you use a For Each loop (or a control that handles the looping for you). For example, you can write the time zone names to the Web page like this by accessing this object’s DisplayName property:

```csharp
For Each tz As TimeZoneInfo In q
    Response.Write(tz.DisplayName & "<br />")
Next
```

A GridView control is handy for looking at what’s inside some objects. The following picks up the bindable properties and values found in q:

```csharp
GridView1.DataSource = q
GridView1.DataBind()
```

A query must begin with a From clause. One reason for the placement is that IntelliSense knows what you’re digging into and can pop up the appropriate objects, methods, and properties for the thingy’s type.

Narrowing the thingies with a Select clause

A Select clause helps you specify exactly what you want returned from a query. As you see in the preceding From examples, LINQ doesn’t require you to use Select. If Select is missing, LINQ creates Select, behind the scenes, as it were. For example, to the compiler, this query
Dim q = From f In System.Drawing.FontFamily.Families

is identical to this query:

Dim q = From f In System.Drawing.FontFamily.Families _
    Select f

You can test the result with this:

For Each fnt In q
    Response.Write(fnt.Name & "<br />")
Next

Both of the preceding queries return a collection of System.Drawing.FontFamily objects. In plain English, the code tells .NET, “Here’s an iteration variable named f. Take it with you and examine each item inside Families. As for what to select, well, just bring back all the f — er, font — thingies.”

Bringing back every property and embedded object might be what you want. Perhaps you want only the names and not the complete objects. If so, you can tell .NET to bring back whatever’s in the Name property:

Dim q = From f In System.Drawing.FontFamily.Families _
    Select f.Name
For Each fnt As String In q
    Response.Write(fnt & "<br />")
Next

Notice that in the preceding For Each loop, you can use As String because the query is selecting the Name property, which is a String type.

**Filtering with a Where clause**

Using a Where clause has tremendous power because Where decides what gets into the result. Here’s an interesting query that looks at the memory usage of whatever’s running on the Web server:

Dim q = From p In _
    Select p.Id, p.ProcessName, p.PrivateMemorySize64 _
    Where p.PrivateMemorySize64 > 1000000
GridView1.DataSource = q
GridView1.DataBind()
The preceding Select clause tells .NET to return the `Id`, `ProcessName`, and `PrivateMemorySize64` properties from each object (represented by the iteration variable `p`).

A Where clause filters the results by saying, “Return only those processes that use more than one million bytes of memory.” However, a careful look at the Where clause shows that it’s using `PrivateMemorySize64` for the filter, not `p.PrivateMemorySize64`. Why not include the iteration variable?

In LINQ queries, it’s all about scope — what’s in and what’s out. In this case, the Select clause narrows the scope to the three properties. And the `p` variable? As a mobster would say, “Fuggedaboutit!” The `p` has gone out of scope and can no longer be used. After the Select clause in this example, the Where clause can access only `Id`, `ProcessName`, and `PrivateMemorySize64`.

Here’s a version of the memory snooper with the Where clause first:

```vbnet
Dim q = From p In _
    Where p.PrivateMemorySize64 > 1000000 _
    Select p.Id, p.ProcessName, p.PrivateMemorySize64
```

This time, you use the iteration variable (`p`) to filter the results because that’s what is in scope. Nothing in the Where clause reduces what’s available.

If your queries aren’t working and IntelliSense isn’t cooperating with the member names from the iteration variable, check whether a Select clause has taken the iteration variable out of scope.

**Filtering with an Eye on Strings**

Visual Basic gives you several ways to filter the results of a query based on what’s in a string of text. This section shows the use of operators and methods, including `Like`, `Contains()`, `EndsWith()`, and `StartsWith()`.

**Choosing what you Like**

The follow code examines the ASP.NET `ServerVariables` collection in the Request object and uses the `Like` keyword to look for the text `SERVER`. 
Dim q = From sv As String In Request.ServerVariables _
  Select sv _
  Where sv Like "*SERVER*"
GridView1.DataSource = q
GridView1.DataBind()

The asterisks on both ends of *SERVER* indicate that any number of characters (or none) can exist before the string as well as after the string. The query returns nine members of the collection. The following partial results show that the location of SERVER within the text isn’t important. What matters is that the string you’re testing matches the pattern:

CERT_SERVER_ISSUER
....
SERVERSOFTWARE

In the preceding example, if you drop the first asterisk — as in Where sv Like "SERVER*" — the query returns all items starting with SERVER.

Investigating what the query Contains() 

The Contains() function is similar to Like in that it allows you to look inside a string and return the items that match. In the following code, Contains() examines each ProductName value and returns those that have the letters co within them, such as Chocolade and Ipoh Coffee.

Dim dc As New NWDataClassesDataContext
Dim q = From p In dc.Products _
  Select p _
  Where p.ProductName.Contains("co")

If your Visual Web Developer complains about the first line of the preceding snippet, make sure that you went through the setup at the beginning of this chapter. That’s where you configure the DataContext object (NWData ClassesDataContext) to use the Northwind database.

It all StartsWith() and EndsWith() strings

The StartsWith() and EndsWith() functions test exactly what they say — whether a string matches the beginning and ending characters, respectively. In geekspeak, this is wildcard matching. Go wild!
The following query uses **StartsWith()** and **EndsWith()** to return names from a string array. The Boolean operator **Or** ensures that both functions contribute to the outcome. For more on logic operators like **Or**, see Chapter 14.

```vbnet
Dim names As String() = _
    {"Elaine", "Brenda", "Julie", "Jaclyn"}
Dim q = _
    From s In names _
    Where _
        s.StartsWith("j", _
            StringComparison.CurrentCultureIgnoreCase) _
        Or _
        s.EndsWith("e", _
            StringComparison.CurrentCultureIgnoreCase)
BulletedList1.DataSource = q
BulletedList1.DataBind()
```

The preceding displays Elaine, Julie, and Jaclyn in an ASP.NET Bulleted List control. Notice that Julie fits both tests (because her name starts with j and ends with e), but her item appears only once in the result. This test ignores the letter case so that J and j are equivalent.

### Filtering Based on Numbers

To return results based on numerical values, use the standard comparison operators, such as <, <=, >, >=, <>, and =. This section includes some basic techniques for filtering data by testing numbers.

### Finding expensive items

In the following snippet, the query gets the name and price of all products that cost more than 30 dollars. Notice that the **Select** clause in this example asks for two properties from the **Products** object. Therefore, the object itself is not included and is no longer available to the **Where** clause:

```vbnet
Dim dc As New NWDataClassesDataContext
Dim q = From p In dc.Products _
    Select p.ProductName, p.UnitPrice _
    Where p.UnitPrice > 30
GridView1.DataSource = q
GridView1.DataBind()
```

The preceding query uses the **DataContext** classes that you create at the beginning of this chapter.
Filtering dates and times

Dates are tricky in .NET because the date format depends on the operating system's language and culture settings. In this example, you set the culture in code to be sure that ASP.NET parses the text accurately. The next step is to create an array of Date values and apply dates to them. The LINQ part comes in with a query. The Where clause checks that the date (when parsed) is later than January 13, 2008.

```vbnet
Dim cinfo As New System.Globalization.CultureInfo("en-CA")
Dim dt(2) As Date
dt(0) = Date.Parse("December 24, 2007", cinfo)
dt(1) = Date.Parse("May 22, 2008", cinfo)
dt(2) = Date.Parse("February 10, 2008", cinfo)

Dim q = From d In dt Where d > DateTime.ParseExact("January 13, 2008", "D", cinfo) Select d.ToLongDateString
CheckBoxList1.DataSource = q
CheckBoxList1.DataBind()
```

The Select clause not only selects the dates but also formats them nicely like this: Sunday, February 10, 2008.

Visual Basic also accepts dates in a SQL-like format, such as #1/13/2008#, as shown here. Keep in mind that the month/day/year arrangement is prone to misinterpretation and error outside the United States.

```vbnet
Dim q = From d In dt Where d > #1/13/2008# Select d.ToLongDateString
```

Thoroughly Aggregating Data

LINQ queries help you collect data and report on it from several perspectives, such as how many items, total value, average cost, minimum value, and maximum number. This section shows you how to get results by using aggregation.

Just give me the list and the Count() 

You can get the number of rows of data or items in an array by using the extension method Count(). Just tack Count() onto the end of a collection or query result, and you’re done! As shown in Figure 7-2, this example gets a list of .aspx files from the Web site’s root directory.
The following code displays the count in a Label control, and the file details (abbreviated in Figure 7-2) in a GridView control:

```vbnet
Try
    Dim q = From f In New System.IO.DirectoryInfo _
        (Server.MapPath("~")).GetFiles() _
        Select f _
        Where f.Extension = ".aspx"
    Label1.Text = "ASPX count: " & q.Count().ToString()
    GridView1.DataSource = q
    GridView1.DataBind()
Catch ex As Exception
    Label1.Text = "Not allowed to do that!"
End Try
```

In the preceding code, you get the Count() directly from the query result (represented by the variable q) and then convert the number to a string for display.

Notice the use of Try...Catch...End Try in the example. Accessing file information on a Web server might result in permissions errors on the Internet host’s system. For more on error handling, see Chapter 21.

**If at first you don’t succeed, you’re running about Average()**

The Average() function takes a set of values, totals them, and then divides by the number of items. This example returns all the products from the Northwind database and selects the UnitPrice property for each. Notice that the query is wrapped in brackets. That way, the Average() function applies to the complete results from the selection:
Dim dc As New NWDataClassesDataContext
Dim q = (From p In dc.Products _
    Select p.UnitPrice).Average()
Labell.Text = "The unit price is " & _
    Format(q, "C")

The last line of code turns the query result (the average) into currency, using the Format() function with C for currency. At runtime, the page displays The unit price is $28.87. (My British colleague Mark Rae reminds me that he sees £ rather than a $ for the currency symbol.)

**First the Dim and then the Sum()**

Count() tells you how many items the query returns, and Sum() tells you the total of all the item values. In the following snippet, the From clause looks at the contents of the Products table, and the Select clause returns the UnitPrice property for each item. The Sum() function adds the Unit Price values:

Dim dc As New NWDataClassesDataContext
Dim q = (From p In dc.Products _
    Select p.UnitPrice).Sum()
Labell.Text = "The total is " & _
    Format(q, "C")

A US and Canadian server displays the text The total is $2,222.71.

**Returning the Min() and the Max() values**

Instead of jumping through hoops (or loops) to find the largest and smallest values in a collection, you can call the Min() and Max() extensions methods. In this example, the query selects the unit price values of all products and puts the results in the variable q. After q holds an array, extracting the highest and lowest unit prices is easy:

Dim dc As New NWDataClassesDataContext
Dim q = From p In dc.Products _
    Select p.UnitPrice
Labell.Text = "The cheapest item is " & _
    Format(q.Min, "C") & _
    " and the most expensive is " & _
    Format(q.Max, "C")
The result of selecting the UnitPrice values, applying Min() and Max(), and formatting the result is the following text: The cheapest item is $2.50 and the most expensive is $263.50.

**Stepping along with Skip() and Take()**

You often want to display a large number of data items in chunks. The Skip() and Take() operators help you page or step through a list.

The Skip() operator does just what it says: It looks at a list and jumps ahead to the specified position. For example, if you call Skip(10) in a LINQ query, you land on the tenth item. What you do when you arrive is your business, but a common action is to Take() something.

The Take() operator starts from the current point and grabs the given number of items. Take(5) takes a break after gathering the next five items (if there are five to take). Figure 7-3 shows a GridView control that you create in the following steps. It displays five items each time you click the button.

To use Skip() and Take() for viewing small chunks of data in a GridView, follow these steps:

1. Add an ASP.NET Web form named sandt.aspx to your project.
2. From the Toolbox, drop a GridView control and a Button control on the page.
3. In Design view, double-click a blank area of the page to create a default handler for the Page Load event and then add the following statements inside the subroutine:
   ```vbscript
   If Not IsPostBack Then
       GetData()
   End If
   ```
4. In Design view, double-click the Button control and then add this statement to the Button1_Click() handler subroutine:
   ```vbscript
   GetData()
   ```
5. **In Source view, add the following subroutine:**

```vbnet
Protected Sub GetData()
    Dim intPos As Integer = 0
    If IsNothing(ViewState("pos")) Then
        ViewState("pos") = 0
    Else
        intPos = Convert.ToInt32(ViewState("pos")) + 5
        ViewState("pos") = intPos
    End If

    Dim q = From c In System.Globalization.CultureInfo.GetCultures(2)
            Select c.EnglishName
            Order By EnglishName Skip (intPos) Take (5)
    GridView1.DataSource = q
    GridView1.DataBind()
End Sub
```

At runtime, the page loads and executes the `GetData()` subroutine. On the first run, the routine initializes a ViewState variable called `pos` that tracks the position within the list of items during postbacks. The LINQ query fetches the culture names and puts them in alphabetical order. The `Skip()` operator jumps ahead to the value of `intPos` (which is zero the first time). Finally, `Take(5)` takes five items and quits.

When you click the button, the `GetData()` routine runs again. However, this time, the position counter (`intPos`) is increased by five so that `Skip()` starts five items farther into the list.

### Grouping, Sorting, and Making Distinct

LINQ shines when you need to put data into categories or groups. In this example, you use the `Group By`, `Order By`, and `Distinct` keywords to sort and display information about cultures.

### Creating the language grouping page

Figure 7-4 shows part of the Web page that you create in this section. The categories (such as `el`, `en`, and `et`) are the two-letter language codes as stored by Windows in the `CultureInfo` object. The language and countries appear within their categories. For example, Greek (Greece) appears within `el` and English (Australia) within `en`. Notice that the query results sort the language codes and countries alphabetically. The page uses two ASP.NET `DataList` controls, one embedded in the other.
To create the grouped and sorted language example, follow these steps:

1. **Add an ASP.NET Web form named** `languages.aspx` **to your page.**

2. **In Design view,** double-click a blank area to create a handler for the `Page Load` event and then insert the following code inside the handler routine:

   ```vbc
   Dim q = _
   From c In _
   System.Globalization.CultureInfo.GetCultures(2) _
   Order By c.EnglishName _
   Order By c.TwoLetterISOLanguageName Distinct _
   Group c By c.TwoLetterISOLanguageName Into Group _
   Select New With _
   {.ISO = TwoLetterISOLanguageName, .culture = Group}
   dlLetters.DataSource = q
   dlLetters.DataBind()
   ```

3. **In Source view,** add the following Import directive to the top of the page:

   ```vbc
   <#@ Import Namespace="System.Linq" %>
   ```

4. **In Source view,** add the following markup inside the HTML `<form>` `</form>` tags.
At runtime, the code retrieves the data and assigns it to the out DataList control to create the page in Figure 7-4. The next topic explains the grouping technique line by line.

### Analyzing the LINQ grouping query

The LINQ query used here gets its data from one source but uses the same data for two purposes. The first part of the query sets the source for the raw data: the `System.Globalization.CultureInfo` object. Its `GetCultures()` method returns a collection containing all cultures known to the Windows operating system. The iteration variable `c` represents the individual `CultureInfo` objects. The `From` clause uses `e` as it passes through the collection:

```csharp
Dim q = _
    From c In _
    System.Globalization.CultureInfo.GetCultures(2) _
```

Next, the `Order By` clause sorts the `CultureInfo` objects according to the `EnglishName` property.

A second `Order By` clause sorts the `CultureInfo` objects according to the `TwoLetterISOLanguageName` property and uses the `Distinct` keyword to eliminate duplicates.

Norway, Sweden, and Finland use three-letter codes as their `TwoLetterISOLanguageName`.

```csharp
Order By c.TwoLetterISOLanguageName Distinct _
```
The next section creates groups of languages, such as el, en, and es. The `Group By` clause takes the collection of `CultureInfo` objects and groups them according to the `TwoLetterISOLanguageName` property (such as en). That arrangement is set aside by storing it temporarily in the `Group` keyword that acts somewhat like a variable:

```csharp
Group c By c.TwoLetterISOLanguageName Into Group _
```

The code that has gone before leaves you with a list full of `TwoLetterISOLanguageName` values and a collection of `CultureInfo` objects arranged into the `Group` keyword.

You access the values by using `Select` and custom collections. In this case, you put the sorted list of two-letter names into a variable called `ISO` and the collection of grouped `CultureInfo` objects into a variable called `culture`.

In Visual Basic, you use the `New With` keywords to create these objects with LINQ. The naming convention requires a dot (.) before the variable name:

```vbnet
Select New With _
{.ISO = TwoLetterISOLanguageName, .culture = Group}
```

By this point, you have a set of ISO names and a collection of culture objects. To display the information on the page, you need to bind the data to a suitable control — in this case, a `DataList` control named `dlLetters`:

```csharp
dlLetters.DataSource = q
dlLetters.DataBind()
```

## Rendering grouped data on a Web page

The preceding code assigns the result of the query to the `DataList` control named `dlLetters`, but it doesn’t render all the data on the Web page. The outer `DataList` control uses the `Eval()` method to set the `Text` property of an embedded `Label` control. This displays all the two-letter language names found in the `ISO` array that’s inside the variable `q`:

```xml
<asp:DataList ID="dlLetters" runat="server">
    <ItemTemplate>
        <asp:Label ID="lblISO" runat="server" 
            Text='<%# Eval("ISO") %>'>
        </asp:Label><br />
    </ItemTemplate>
</asp:DataList>
```

Just like the preceding `Label` control (`lblISO`) can bind to its container’s data (the `DataList` named `dlLetters`), a `DataList` can bind to its container’s data. That’s why you can embed another `DataList` control within the `ItemTemplate` block. This `DataList` binds to the `culture` object
belonging to its container. The Label control (lblName) gets its data from the culture object’s EnglishName property, using the Eval() method as shown here:

```xml
<asp:DataList ID="DataList1" runat="server"
    DataSource='<%# Eval("culture") %>'>
    <ItemTemplate>
        <asp:Label ID="lblName" runat="server"
            Text='<%# Eval("EnglishName") %>'><br />
        </asp:Label><br />
    </ItemTemplate>
</asp:DataList>
```

To group data using LINQ to SQL, see the section on displaying hierarchical data with LINQ in the next chapter.

### Using LINQ to Create and Query XML

LINQ to XML uses LINQ syntax to generate, read, and sort XML data. LINQ to XML uses its own set of objects, such as XDocument, XElement, and XAttribute to represent parts of an XML document. It isn’t quite as straightforward as regular LINQ, but as you can see in the next few paragraphs, the basic techniques are the same.

In this section, you build a class that holds information about family members (nieces and nephews). Next, you create objects from the class and configure the object properties.

The objects become the data source for an XML document that you save to a file on the Web server. In the final phase, you read the XML file and filter its data via LINQ syntax.

### Creating the KinFolk class

The KinFolk class represents a niece or a nephew. To keep the code short, you record only the person’s first name, gender, and father’s name.

The KinFolk class uses the private variables _fname, _gender, and _father as well as the corresponding public properties Fname, Gender, and Father. To create the KinFolk class, follow these steps:

1. Add a class file named kinfolk.vb to the App_Code folder of your project (File ➔ New File ➔ Class ➔ Add).
2. Use contents of Listing 7-1 as the complete contents of kinfolk.vb.
The next task is to use the class to create objects.

Listing 7-1: The KinFolk Class to Represent Nieces and Nephews

```vbnet
Public Class KinFolk
    Private _fname As String
    Private _gender As String
    Private _father As String

    Public Property FName() As String
        Get
            Return _fname
        End Get
        Set(ByVal value As String)
            _fname = value
        End Set
    End Property

    Public Property Gender() As String
        Get
            Return _gender
        End Get
        Set(ByVal value As String)
            _gender = value
        End Set
    End Property

    Public Property Father() As String
        Get
            Return _father
        End Get
        Set(ByVal value As String)
            _father = value
        End Set
    End Property

End Class
```

Using object initializers to add data

The preceding KinFolk class sets the structure that allows you to create objects (instantiate). In this section, you use variable `nn` to reference a generic list of KinFolk objects and then fill the objects with data.

Using object initializers is an easy, shorthand way of creating an object and adding data in one line of code. As shown in Listing 7-2, you precede the property name (for example, `Father`) with a dot (.), followed by an equal sign (=) and the value to assign.

To build an ASP.NET page that generates the objects, follow these steps:
1. In your Web project, add an ASP.NET Web form named createxml.aspx.

2. In Design view, add an ASP.NET Label control to the page.

3. In Source view, within the `<script runat="server"></script>` tags, add the CreateData() subroutine, as shown in Listing 7-2.

After you have the data in objects, you have something that you can write to an XML file. That part comes next.

Listing 7-2: Using CreateData() to Build a List of KinFolk Objects

```vbnet
Public Function CreateData() As List(Of KinFolk)
    Dim nn As New List(Of KinFolk)
    nn.Add(New KinFolk With {.FName = "Dave", .Father = "Ron", .Gender = "m"})
    nn.Add(New KinFolk With {.FName = "Karen", .Father = "Ron", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Amy", .Father = "Mike", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Meghann", .Father = "Mike", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Holly", .Father = "Mike", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Kurtis", .Father = "Paul", .Gender = "m"})
    nn.Add(New KinFolk With {.FName = "Rachel", .Father = "Paul", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Elaine", .Father = "Stan", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Brenda", .Father = "Stan", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Julie", .Father = "Stan", .Gender = "f"})
    nn.Add(New KinFolk With {.FName = "Jaclyn", .Father = "Stan", .Gender = "f"})
    Return nn
End Function
```

Building the XML file with LINQ to XML

LINQ to XML can generate a complete, well-formed XML file based on almost any data you provide. In this example, you use the list of KinFolk objects from Listing 7-2. It could just as easily be rows of data from a database. Follow these steps to read the data, build an XML document, and save the XML to a file:

1. In createxml.aspx (added in the preceding section), switch to Design view and double-click an empty area of the page to create an event handler for the Page Load event.

2. Inside the Page_Load() subroutine, add the code in Listing 7-3.

3. At the top of the source code, add the following directives:

   ```xml
   <%@ Import Namespace="System.Collections.Generic" %>  
   <%@ Import Namespace="System.Xml.Linq" %>
   ```

4. If your project doesn’t already have an App_Data folder, create one. (Right-click the project name and then choose Add ASP.NET Folder: App_Data.)
5. **Browse to** createxml.aspx.

A new file named niecesandnephews.xml appears in the App_Data folder. (You may need to refresh Solution Explorer to see the new file.) Here’s a sample of the XML file’s contents:

```xml
<kinfolk>
  <kin>
    <fname gender="m">Dave</fname>
    <father>Ron</father>
  </kin>
</kinfolk>
```

Listing 7-3:  Building an XML Document with LINQ to XML

```csharp
Dim nn = CreateData()
Dim kfdoc As New XDocument(New XDeclaration("1.0", Nothing, Nothing))
kfdoc.Add(New XComment("LINQ to XML Generated File"))
Dim kfrootelement As New XElement("kinfolk")
Dim kfitem As XElement
Dim kffirstname As XElement
Dim kffather As XElement
Dim kfgender As XAttribute
For Each n In nn
    kfitem = New XElement("kin")
    kffirstname = New XElement("fname", n.FName)
    kfgender = New XAttribute("gender", n.Gender)
    kffirstname.Add(kfgender)
    kfitem.Add(kffirstname)
    kffather = New XElement("father", n.Father)
    kfitem.Add(kffather)
    kfrootelement.Add(kfitem)
Next
kfdoc.Add(kfrootelement)
Try
    kfdoc.Save(Server.MapPath("~/App_Data/niecesandnephews.xml"))
    Label1.Text = "File written"
Catch exc As Exception
End Try
```

Here’s how Listing 7-3 works:

→1  The **CreateData()** subroutine gets the data to use in the XML file.

→2-3  The variable **kfdoc** holds a **new XDocument object** that includes an XML declaration created with an **XDeclaration object**.

→4  The **kfrootelement** establishes the **root node** of the XML (which renders as `<kinfolk>` in the XML file).

→5-8  The **Dim** statements declare several variables as **XElement types** so they can be used in the **For Each** loop, where most of the work takes place.
The For Each loop starts working its way through the list of KinFolk objects as represented by variable nn. On each pass through the loop, the code creates a <kin> element and adds an <fname> element with the value held in the FName property of the object. Notice that using the XAttribute object creates a gender attribute that goes becomes part of the <fname> element by calling kfirstname.Add().

After creating the <father> element and including its value from n.Father, the entire node is added to the parent element, which in turn is added to the root element.

The XDocument object’s Save() method tries to save the XML file. This part is wrapped in a Try...Catch...End Try sequence so the page doesn’t crash if the ASP.NET account doesn’t have write permission in the destination folder.

Filtering XML with a LINQ to XML query

The familiar LINQ syntax lets you query and filter the contents of an XML file. In this example, you use niecesandnephews.xml generated in the preceding section. In this case, the goal is to obtain a list of nieces whose father is Ron or Stan. What’s more, the names should appear in alphabetical order. To filter an XML document with LINQ to XML, follow these steps:

1. Add an ASP.NET Web form named nandn.aspx to your project.
2. In Source view, add the following Import directives to the top of the page:
   ```
   <%@ Import Namespace="System.Linq" %>
   <%@ Import Namespace="System.XML.Linq" %>
   
   Dim root As XElement = _
   XElement.Load(Server.MapPath(_
   ("~/App_Data/niecesandnephews.xml"))
   Dim q = From k In root.Descendants _
   Where (k.Element("father") = "Ron" Or _
       k.Element("father") = "Stan") _
   Where k.Element("fname").Attribute("gender") = "f" _
   Select First_Name = k.Element("fname").Value, _
       Father = k.Element("father").Value _
   Order By First_Name
   
   GridView1.DataSource = q
   GridView1.DataBind()
   ```
At runtime, the GridView control looks like Figure 7-5.

<table>
<thead>
<tr>
<th>Father</th>
<th>First_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stan</td>
<td>Brenda</td>
</tr>
<tr>
<td>Stan</td>
<td>Elaine</td>
</tr>
<tr>
<td>Stan</td>
<td>Jaclyn</td>
</tr>
<tr>
<td>Stan</td>
<td>Julie</td>
</tr>
<tr>
<td>Ron</td>
<td>Karen</td>
</tr>
</tbody>
</table>

The code in Step 4 uses the XElement object’s Load() method to get the XML content from niecesandnephews.xml and stores it in the root variable. In this case, the root is the <kinfolk> element. The data you want to query is within each <kin> element, so the query starts looking through the collection root.Descendants.

As the iteration variable (k) looks at each item in the collection, the Where clause checks whether the <father> element’s value is Ron or Stan. This query has an additional Where clause that looks into the <fname> element’s gender attribute to find those with the value f. (Remember: You’re looking for nieces.) The Select clause chooses two parts of a matching item: the fname element’s value (using the alias First_Name) and the father element’s value (using the alias Father). Finally, the Order By clause tells the query to sort everything alphabetically by the niece’s first name.

The last two lines of the subroutine tell GridView1 to use the result of the query as its data source and to bind to that data immediately.

The next chapter uses many of the same techniques to dig into data from SQL Server.
Chapter 8

Using LINQ to SQL and the LinqDataSource

In This Chapter

- Using the object relational designer
- Filtering data in LinqDataSource
- Understanding LINQ to SQL syntax
- Grouping and displaying hierarchical data
- Updating and inserting with DataContext
- Creating a user interface with the ListView control

In Chapters 3 and 4, I show you how to build a database and an ASP.NET page to carry out basic CRUD functions (Create, Retrieve, Update, and Delete). This chapter takes Rapid Action Development (RAD) to a further level by using the Microsoft LINQ to SQL technology and the LinqDataSource control.

In LINQ to SQL, the object relational designer analyzes an ordinary SQL Server database and presents the data tables and stored procedures to code as objects. Visual Web Developer has a good understanding of what goes into objects, including properties, collections, and methods. With its knowledge of objects, Visual Web Developer can help with the syntax while you write code and warn you of problems at design-time.

Building a LINQ to SQL CRUD Page

The high priests of geekdom usually frown on RAD and its associated design tools. Perhaps they invoice clients by the hour, enjoy typing, and have no reason to get home on time. For the rest of us, though, it makes sense to use the best tools to build a very useful, data-driven page, like I show you how to do here.
This chapter requires using the Microsoft Northwind database with SQL Server or SQL Server Express. Chapter 6 includes details on where you can download the Northwind files and how to connect via Visual Web Developer.

Creating the database access code

Although creating the database access code suggests that you’re going to create code, the meaning is akin to a worker who steers the paving machine “paves” the Alsace road. The tool does the hard work while you admire your efforts. Follow these steps to use the object relational designer to generate the database access code that you use throughout this chapter:

1. Add the Northwind database to your project (see Chapter 6).

2. Add a LINQ to SQL Classes file called NWDataClasses.dbml to your project (File ➪ New File ➪ LINQ to SQL Classes ➪ Add).

   If you see a prompt to put the file in the App_Code folder, click Yes.

   The split-page, object relational designer appears.

3. From Database Explorer (Server Explorer in non-Express versions), expand the Tables node.

4. As shown in Figure 8-1, drag the Categories, Order Details, Products, and Suppliers table names from the Database (Server) Explorer and drop them on the left-hand (larger) design surface of NWDataClasses.dbml.

   The designer creates objects based on the table names. For example, the Categories table becomes the Category object, which is stored in NWDataClasses.designer.vb. The database access code is finished in seconds.

Figure 8-1: The object relational designer with Northwind tables.
Hooking up with the LinqDataSource control

A LinqDataSource control acts as an intermediary between the database access code that you create in the preceding section and the data-bound ASP.NET control that you add later in this chapter. The LinqDataSource control resides in the .aspx page as declarative markup, which means that you can configure it in Design view by using its Tasks menu, or in Source view by typing properties and values.

Follow these steps to connect the LinqDataSource with the classes that represent the database:

1. Add an ASP.NET page named products.aspx to your project.
2. In Design view, from the Toolbox, drag a LinqDataSource control and drop it on the ASP.NET page.
3. From the Tasks menu of the LinqDataSource control, choose Configure Data Source.
4. In the Choose a Context Object window, select NWDataClassesDataContext and then click Next.
5. In the Configure Data Selection window, in the Table area, select Products (Table<Product>) from the drop-down list.
6. In the Select area, make sure the asterisk (*) is checked.
7. Click the Advanced button (right-hand side), select all the advanced options, click OK, and then click Finish.

The preceding steps generate the following declarative markup:

```xml
<asp:LinqDataSource ID="LinqDataSource1" runat="server" ContextTypeName="NWDataClassesDataContext" EnableDelete="True" EnableInsert="True" EnableUpdate="True" TableName="Products">
</asp:LinqDataSource>
```

The ContextTypeName property points to the classes you created, and the TableName property refers to the Products table of the database. You’re now ready to work at the user interface level.

Creating the user interface with a ListView

A ListView control generates a grid on which users can view, insert, delete, and update data. It’s a templated control, which means that you can design almost everything that goes into it. See Chapter 13 to read about designing
ListView and other templated controls. In this example, you let a designer generate the layout based on what the LinqDataSource control discovers in the data classes. Follow these steps to create the user interface:

1. **From the Toolbox, add a ListView control to the page.**
2. **From the Tasks menu (use the Smart Tag), choose the data source, LinqDataSource1.**
3. **From the ListView Tasks menu, choose Configure ListView.**
4. **In the Configure ListView window, select Grid, Professional, and all four options. Then click OK.**

The ListView control's wizard includes three columns that you don’t want to display. Therefore, you need to remove these columns from the markup. The ListView templates have no designer support, so this is a manual cleanup in Source view. To remove the unwanted columns, follow these steps:

1. **Open products.aspx in Source View.**
2. **Remove all the table columns (<td> to </td>) that contain Label and TextBox controls that use these IDs:**
   
<table>
<thead>
<tr>
<th>Label/TextBox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order_DetailsLabel</td>
</tr>
<tr>
<td>CategoryLabel</td>
</tr>
<tr>
<td>SupplierLabel</td>
</tr>
<tr>
<td>Order_DetailsTextBox</td>
</tr>
<tr>
<td>CategoryTextBox</td>
</tr>
<tr>
<td>SupplierTextBox</td>
</tr>
</tbody>
</table>

3. **Remove the column headers (<th> to </th>) that contain the text**
   
   Order_Details, Category, and Supplier.

After customizing the ListView control, don’t use the Refresh Schema or Configure ListView items from the ListView Tasks menu. You can lose all your customizations by reentering this designer.

### Using LINQ to work around a deletion constraint

After all this effort, you certainly deserve to browse to the page, edit the data, and use the navigation. I’ll just wait here until you finish admiring your work — and perhaps discovering a nasty surprise.

Did you click a Delete button? The page crashes with the taint of a complaint about a constraint. You fix that problem in this section.

The designers of the Northwind database didn’t want anyone to mess up the data integrity, so a database rule — a *constraint* — was intentionally added.
Why? Say you’re allowed to delete product ID 54 (Tourtière). What happens to the orders that refer to product ID 54 that suddenly no longer exists? A constraint throws an error at anyone who wants to delete a product with a message like, “Go away! There are still references to this product!” Or something like that.

The workaround (at least for this example) is to abide by the constraint: remove all the references to the product first and then delete the product. True, this technique strips data and ruins the historical analysis of company sales, but chalk it up to a learning experience.

A LinqDataSource control lets you know that it intends to delete something by raising a Deleting event before actually deleting. This event interval lets your code intervene (if necessary) or even stop the deletion. In this case, you use the event handler to remove all references to the product so you can then delete the product itself without an error. Follow these steps to insert code into the Deleting event handler:

1. In Source view, put the cursor inside the <script></script> tags.
2. In the upper-left of the editing pane, select LinqDataSource1 from the drop-down list.
3. From the upper-right of the editing pane, select Deleting from the drop-down list.

This creates a skeleton handler routine for the Deleting event that looks like the following (reformatted with line breaks):

```
Protected Sub LinqDataSource1_Deleting _
(ByVal sender As Object, _
ByVal e As System.Web.UI.WebControls._
LinqDataSourceDeleteEventArgs)
End Sub
```

It also adds a reference to the routine inside the markup for LinqDataSource1 like this:

```
OnDeleting="LinqDataSource1_Deleting"
```

4. Insert the following code inside the handler (that is, above the End Sub statement):

```
Dim dc As New NWDataClassesDataContext
Dim prod As Product
prod = CType(e.OriginalObject, Product)
Dim q = From o In dc.Order_Details _
Where o.ProductID = prod.ProductID
For Each od As Order_Detail In q
    dc.Order_Details.DeleteOnSubmit(od)
Next
dc.SubmitChanges()
```
The code in Step 4 starts by using the variable `dc` to get a reference to the data context object that you create in the earlier section, “Creating the database access code.” The next line creates a `prod` variable to hold a `Product` object. A `prod` represents a single product from the Products table of the database.

Notice the use of `e.OriginalObject`? The `e` is a `LinqDataSourceDeleteEventArgs` object that contains a reference to the item that’s about to be deleted. To ensure that .NET knows that this is a `Product` object, use the ` CType()` method to “cast” the object as a `Product`.

The variable `q` represents the results of a LINQ query. Here’s how the query instruction sounds in plain English:

```
Using o as a placeholder (iteration variable to geeks), dig into the object that represents the Order_Details data table. As you poke around, put each item into the placeholder (o) temporarily so you can get a closer look at its properties, especially the ProductID value. Whenever you come across an Orders ProductID value that’s the same as a Products ProductID, hang on to that item!
```

At this point, the variable `q` holds a collection of objects that have the same ProductID. And what does a geek do with collections? (Monty Python voice here.) They iterate collections!

The `For Each` loop (shown again here) uses the variable `od` as a placeholder for each `Order_Detail` object while it works its way through the collection represented by the variable `q`. On each pass through the loop, the code calls the `DeleteOnSubmit()` method to mark the object for removal.

```
For Each od As Order_Detail In q
    dc.Order_Details.DeleteOnSubmit(od)
Next
```

Calling `DeleteOnSubmit()` only flags the items for deletion: It doesn’t actually delete the row. You must call the `SubmitChanges()` method to formally tell the database to proceed with the deletions.

**Confirming deletion requests**

The downside of creating deletion code in the previous section is that clicking the Delete button starts to, er, delete with no questions asked. When a user is about to do something irreversible, it’s polite to confirm that the click wasn’t just a slip of the mouse. Follow these steps to create a confirmation prompt:
1. In Source view, locate the `<ItemTemplate>` tag — and, within it, the button with the ID of DeleteButton.

2. Add the following attribute/value pair (on a single line without a line break) into the Button’s markup:

```html
OnClientClick="return confirm('Do you really want to remove this product and all references to it in the Order Details table?');"
```

3. Locate the `<AlternatingItemTemplate>` element and the second DeleteButton and add the preceding code in this button as well.

The code in Step 2 executes the client-side JavaScript `confirm()` method when the user clicks a Delete button. (Note two Delete buttons: one in the regular item template, and the other in the alternating template.) If the user clicks the prompt’s Cancel button, `confirm()` returns a value of false that prevents the server-side `Click` event from firing.

**Enhancing Usability with LinqDataSource**

The grid in the preceding section handles adding, editing, and deleting products. However, it’s not friendly. In this section, you replace the supplier number with the company name to make it easier for users to add products.

**Putting a name to a number**

At runtime, the ListView you create in the preceding section shows SupplierID and CategoryID values, which are numbers. Instead of asking viewers to match the SupplierID to the company name, you can do it for them. Figure 8-2 shows that NWDataClasses.dbml includes a Supplier object with a CompanyName property. Both the company and category names are available to display in the ListView control.
Follow these steps to use the name of the supplier in the SupplierID column:

1. Open \texttt{products.aspx} \textbf{in Source view}.
2. Open the Quick Replace window (Edit$\Rightarrow$Find and Replace$\Rightarrow$Quick Replace).
3. Set \texttt{Eval("SupplierID")} as the Find What text and \texttt{Eval ("Supplier.CompanyName")} as the Replace With text in the current document.
4. Click Replace All.
   Three occurrences are replaced.
5. Set \texttt{Eval("CategoryID")} as the Find What text and \texttt{Eval ("Category.CategoryName")} as the Replace With text in the current document.
   Three occurrences are replaced.
6. Close the Find and Replace window.

With this change, you can display the name of the category instead of the CategoryID number by using \texttt{Eval("Category.CategoryName")}.

\begin{center}
\textbf{Allowing users to select from a drop-down list}
\end{center}

When users revise or add a product, they’d rather select a recognizable supplier name (such as Ma Maison; see Figure 8-3) than enter its meaningless SupplierID number. You fix that here.

\begin{center}
\textbf{Adding and configuring another LinqDataSource}
\end{center}

This time, you use a LinqDataSource to fetch the list of names from the Suppliers table. Follow these steps to enhance the ListView by creating a drop-down list of supplier names:

1. Open the \texttt{previous} \texttt{products.aspx} page \textbf{in Design view}.
2. From the Toolbox, add a LinqDataSource control and set its ID to LinqSupplierDS.
3. Open the LinqDataSource Tasks menu and click Configure Data Source.
4. In the Choose a Context Object screen, select NWDataClassesData Context and then click Next.
5. In the Configure Data Selection screen, from the Table drop-down list, select Suppliers (Table\textless Supplier\textgreater).
6. In the Select area, check SupplierID and CompanyName.

7. Click the OrderBy button, select CompanyName from the Sort By drop-down list, and then click OK.

8. Click Finish.

The LinqDataSource control (named LinqSupplierDS here) selects the list of suppliers from the Suppliers table. Next, you use that data in the ASP.NET DropDownList control.

**Use the LinqDataSource in a DropDownList control**

The drop-down list should display the company name but provide the supplier’s ID number to the database. Therefore, you use the SupplierID value behind the scenes. Follow these steps to add and configure the drop-down list in the InsertItemTemplate area.

1. In Source view, in the `<EditItemTemplate>` section, replace the TextBox code that looks like this:

```xml
<asp:TextBox ID="SupplierIDTextBox" runat="server" Text='<%# Bind("SupplierID") %>'/>
```

with this DropDownList code:

```xml
<asp:DropDownList ID="SupplierIDDropDownList" runat="server" DataSourceID="LinqSupplierDS" DataValueField="SupplierID" DataTextField="CompanyName" SelectedValue='<%# Bind("SupplierID") %>' />
```
2. In the `<InsertItemTemplate>` section, replace the `TextBox` code that looks like this:

```xml
<asp:TextBox ID="SupplierIDTextBox" runat="server" Text='<%# Bind("SupplierID") %>' />
```

with this `DropDownList` code:

```xml
<asp:DropDownList ID="SupplierIDDDL" runat="server" SelectedValue='<%# Bind("SupplierID") %>'/>
</asp:DropDownList>
```

3. In the code area, between the `<script runat="server">` tags, add the following subroutine:

```csharp
protected Sub ListView1_ItemCreated _
(ByVal sender As Object, ByVal e As System.Web.UI.WebControls._
ListViewItemEventArgs) Handles ListView1.ItemCreated

Dim li As ListViewItem
Dim ddl As DropDownList
If e.Item.ItemType = ListViewItemType.InsertItem Then
    li = e.Item
    ddl = li.FindControl("SupplierIDDDL")
    If Not ddl Is Nothing Then
        Dim dc As New NWDataClassesDataContext
        Dim q = From c In dc.Suppliers
                Select txt = c.CompanyName, valu = c.SupplierID
                Order By txt
        Dim itm As ListItem
        ddl.Items.Clear()
        For Each i In q
            itm = New ListItem
            itm.Text = i.txt
            itm.Value = i.valu
            ddl.Items.Add(itm)
        Next
    End If
End If
End Sub
```

You can now browse to the page, click the Edit button for a row, and change the supplier via the drop-down list. Likewise, you can choose the supplier when you insert a product.

The code in Step 3 goes into action when the ListView control creates the type of row that inserts items. The `FindControl()` method gets a reference to the drop-down list control you insert in Step 2. A LINQ query returns a sorted list of supplier names and IDs into variable `q`. A `For Each` loop passes over each of the suppliers and creates a drop-down list item. The loop uses the `Add()` method to insert the items into the drop-down list.
Filtering Data with LinqDataSource

Each product in the Northwind database belongs to a category, such as Beverages and Seafood. You can instruct the LinqDataSource to return only the products within a category. In this section, you create a drop-down list so the user can view a category of products.

Creating a LinqDataSource to fetch categories

This section uses the Category object in the NWDataClassesDataContext classes generated in the earlier section, “Creating the database access code.” You can see that the Category object exists by opening NWDataClasses.dbml. Follow these steps to obtain a list of product categories:

1. Add a LinqDataSource control named LinqCategoryDs to your page.
2. From the Tasks menu of the LinqDataSource control, choose Configure Data Source.
3. Choose NWDataClassesDataContext from the drop-down list and then click Next.
4. On the Configure Data Selection dialog box, from the Table drop-down list, select Categories (Table <Category>).
5. Click the Order By button, select CategoryName in the Sort By area, and then click OK.
6. Click Finish.

Adding a drop-down list and connecting it to the LinqDataSource

You can use an ASP.NET DropDownList control to display the list of categories retrieved by the LinqDataSource control. Follow these steps to add and configure the drop-down list:

1. Add a DropDownList control named ddlFilter to the page, above the ListView control.
2. From the ddlFilter Tasks menu, check Enable AutoPostBack.
3. From the Tasks menu, choose Choose Data Source.

4. On the Choose a Data Source window, select LinqCategoryDS as the data source, CategoryName as the data field to display, and Category ID as the data field for the value. Then click OK.

If the data fields don’t appear after selecting LinqCategoryDS, click the Refresh Schema link in the lower left to give the snoozing data a little nudge.

The drop-down list now has a source for the list of categories. The next task tells the LinqDataSource to apply the filter.

**Filtering the LinqDataSource with a Where parameter**

The first LinqDataSource in this chapter selects all the rows that it finds in the Products database. To make it choosy, give it one or more parameters. The parameter comes from the DropDownList control that you add in the preceding section. Follow these steps to add filtering to the LinqDataSource control:

1. In Design view, select the main LinqDataSource control (probably named LinqDataSource1) and open its Tasks menu.
2. Click Configure Data Source and then click Next.
   
   The context object is already configured, so you don’t need to change it.
3. On the Configure Data Selection screen, click the Where button.
4. In the Column area (upper left), select CategoryID from the drop-down list.
5. From the Operator area, select == (two equal signs).
6. From the Source drop-down list, select Control.
7. In the Parameter Properties area, for the Control ID, select ddFilter.
8. Click Add.
   
   The Where expression appears in the lower preview area.
9. Click OK and then click Finish.

At runtime, the page loads, and the LinqDataSource control gets the value for its Where parameter from the selected item in the drop-down list, usually Beverages. When you select other categories, the AutoPostBack feature refreshes the screen and the items to display.
Displaying Hierarchical Data with LINQ

A LINQ query can organize data in groups or categories for display in the ASP.NET GridView control. In this section, you create a page like Figure 8-4, in which the products for the Northwind database appear according to their category, such as Beverages.

Grouping with a LINQ query

In this example, you create the LINQ query in code rather than in a LinqDataSource control. Listing 8-1 shows all the code you need to create groups of products according to their categories. The code assumes that you create the DataContext objects as explained in the earlier section, “Creating the database access code.” Follow these steps to create a page to use the query:

1. Add an ASP.NET page named hierar.aspx to your project.
2. In Design view, double-click an empty area of the page to create a handler for the Page Load event.
3. Within the Page_Load() subroutine, add the contents of Listing 8-1.

You create the user interface for hierar.aspx in subsequent steps. First, do a walk-through of Listing 8-1 to understand the query and its groups.
Listing 8-1: Grouping by Categories with LINQ’s Group By

```vbnet
Dim dc As New NWDataClassesDataContext
Dim q = From c In dc.Categories, p In dc.Products Where c.CategoryID = p.CategoryID Group p By c.CategoryName Into Group Select New With {.cgname = CategoryName, .prdcts = Group}
gvCategories.DataSource = q
gvCategories.DataBind()
```

As the page loads, the `Dim dc` statement creates a `NWDataClassesDataContext` object based on the classes generated by the object relational designer. (You can explore the generated code by opening `NWDataClasses.designer.vb` in the `App_Code` folder. That’s the code that makes the Northwind data tables look like .NET objects.)

The LINQ query appears complicated until you break it down into smaller chunks. The first step is to declare the variable `q` that holds the results of the query. Think of `q` as a cargo container that you can load for shipping.

**Point the little gnome to the raw material**

Then comes the `From` statement:

```vbnet
From c In dc.Categories, p In dc.Products
```

Imagine a very dedicated gnome inside the query who carries out instructions. The preceding tells the gnome, “Use the variable `c` to represent an individual item in the set of `Categories` objects, and use the variable `p` to represent an item in the collection of `Products` objects. We want you to test each one of the items in these groups.”

The `c` and `p` variables act like an X-ray machine that allows a good look inside each object in a collection. (In geekspeak, the variables `c` and `p` are *range* or *iteration* variables, and they allow you to *dereference* the objects they represent.)

**Giving specifics as to what you want**

The `Where` keyword gets specific about what you want from these objects. The gnome uses the `c` and the `p` variables to probe inside the item to test the value of the `CategoryID` property:

```vbnet
Where c.CategoryID = p.CategoryID
```
The preceding says, “When you’re looking through a category \((c)\), watch the product \((p)\). When you find a CategoryID value that exists in both Categories and Products, keep those items. Discard the rest that don’t have a match.”

**How do you want that grouped?**

The code now has two piles of products. For every CategoryID in one pile, the same CategoryID exists somewhere in the other pile. The gnome’s next task is to separate the items into groups:

```
Group p By c.CategoryName Into Group _
```

The gnome starts looking through the Categories pile. Say the first item he picks up has a CategoryID of 4. Holding that item in his left hand, he searches through the pile of Products and pulls out all items with a CategoryID of 4. To keep things straight, he paints a banner with the value of the CategoryName property (for example, Dairy Products) and assembles the product items underneath the banner. Eventually, the floor of Grouping Brothers Warehouse has several neat piles of products gathered under banners, such as Confections and Seafood.

**Special orders don’t upset you**

Ordinarily, the gnome would bundle the grouped piles into the cargo container (variable \(q\)) and leave for the day. However, this customer wants custom packaging, using his own objects:

```
Select New With _
{.cgname = CategoryName, _
 .prdcts = Group}
```

The preceding Select statement says, “Create a brand-new object with a string property called cgname and put the CategoryName values into that. Also, create a new collection of Products called prdcts and store the groups in that.”

The final part of the code, repeated here, tells the GridView control to use the contents of \(q\) for its data and to bind to it now. You create the GridView control in the next section.

```
gvCategories.DataSource = q
gvCategories.DataBind()
```

**Creating the outer GridView control**

To display hierarchical data, you embed data-bound ASP.NET controls inside each other by using templates. Follow these steps to create and configure an outer GridView control.
1. In Design view, from the Toolbox, add a GridView control named
   gvCategories to the hierar.aspx ASP.NET page that you created
   previously.

2. From the gvCategories Tasks menu, choose Edit Columns.
   The Fields dialog box opens.

3. In the upper-left area, from the Available fields, select TemplateField;
   then click Add.

4. Set the ShowHeader property to False.

5. Clear (uncheck) Auto-generate fields and then click OK.

### Adding a Label control to display categories

Each category name appears on its own row in the outer grid. To display the
category, you need a Label control. To add the Label control inside the GridView,
follow these steps:

1. From the Tasks menu of gvCategories (the outer Gridview), choose
   Edit Templates and then select ItemTemplate.

2. Drag an ASP.NET Label control and drop it inside the ItemTemplate
   area.

3. From the Tasks menu for Label, choose Edit DataBindings.

4. In the DataBindings dialog box, select the Text property in the upper-
   left area, insert the code expression Eval(“cgname”) in the Custom
   binding area, and then click OK.

   You just told the Label control to extract its data from the cgname field
   of the LINQ query.

If you run the page now, you see a small DataGrid that displays the category
names, as shown in Figure 8-5. The next task is to show the products that
belong to each category.

Figure 8-5:
The outer GridView
showing categories.
Creating the inner GridView control

Every category (such as Beverages) should have one or more products. The easiest way to display the products is in a GridView. Follow these steps to put a GridView inside the existing GridView.

1. Select gvCategories. From its Tasks menu, choose Edit Templates.
2. Select the ItemTemplate.
   In the template, you find the Label control added previously in “Adding a Label control to display categories”.
3. Drag a GridView control from the Toolbox and drop it inside the ItemTemplate, below the Label control.
4. From the inner GridView Tasks menu, choose Edit DataBindings.
5. In the DataBindings window, select the DataSource property and then type the following expression in the Custom Binding area:
   ```
   Eval("prdcts")
   ```
   The preceding tells the inner GridView to get its data from a variable named prdcts.
6. Click OK to close the DataBindings window.

By default, a GridView generates a column for every field that it finds in the data source. When you run the page, you see that the GridView analyzed the prdcts object (which is really a collection of products) and created several columns. If all went well, your page resembles Figure 8-4 but without the style enhancements.

Updating Data with a LINQ Query

LINQ to SQL does more than just select data for viewing: It selects it for updates as well. You use the query to get a subset of the data (maybe a subset with just one item) and then loop through the item(s) to do updates. In this section, you add multiple exclamation marks (!) to some items and view the results.

The follow sections insert strange content (exclamation marks) into your Northwind database. Make sure that you have a clean copy of the database set aside for future use.
Exclaiming with an Extension method

When you type in Code view (see Figure 8-6), you’ve no doubt seen IntelliSense pop up all those interesting functions like ToLower(), ToUpper(), and ToExclaim(). Okay, maybe you haven’t seen ToExclaim() yet in IntelliSense, but you’re about to!

*Extension methods* let you tack on functions of your own. The only catch is that the extension code must be wrapped in a Visual Basic Module, not in the .aspx page itself. Follow these steps to create the ToExclaim() extension method:

1. **In your project’s App_Code folder, add a new text file named extmodule.vb (File ➔ Add New Item ➔ Text File).**
2. **Add the following code into extmodule.vb:**
   ```vbnet
   Imports Microsoft.VisualBasic
   Imports System.Runtime.CompilerServices
   Public Module Extns
   <Extension()> _
   Function ToExclaim(ByVal s As String) As String
       Return s & "!!"
   End Function
   End Module
   ```

   That’s all there is to it! The significant part is the <Extension()> attribute (a *decoration*, in geekspeak) that precedes the function name, ToExclaim(). The function accepts a string (as variable s), adds two exclamation marks (!!), and sends the new string back. (I chose this because you will use the function frequently.)
Building a page to update product data

This section updates several product names in the Northwind database by adding exclamation marks to them. If you haven't yet added the DataContext code for this chapter, see the earlier section, “Creating the database access code,” to generate it.

Follow these steps to implement the update code:

1. Add a page called exclaim.aspx to your project (File ➪ New File ➪ Web Form ➪ Add).

2. In Design view, from the Toolbox, add a GridView control and a Button control to the page. Position the Button control above the GridView for visibility.

3. Double-click a blank area of the page to create a skeleton handler for the Page Load event and add the following line of code:

   ```csharp
   BindData()
   ```

   This is a call to the following subroutine that fills the GridView.

4. After the End Sub of the Page_Load() subroutine, add the BindData() method:

   ```csharp
   Protected Sub BindData()
       Dim dc As New NWDataClassesDataContext
       Dim q = From p In dc.Products
               GridView1.DataSource = q
               GridView1.DataBind()
   End Sub
   ```

5. In Design view, double-click the Button control to create a default handler for the Click event and then insert the following code:

   ```csharp
   Dim dc As New NWDataClassesDataContext
   Dim q = From p In dc.Products
           Where p.ProductName.StartsWith("c")
           Select p
   For Each prod In q
       prod.ProductName = prod.ProductName.ToExclaim()
   Next
   dc.SubmitChanges()
   BindData()
   ```

6. Run the page, noting the names of the items that start with C.

7. Click the button at the top of the page.

   Chai appears, with two exclamation marks, as in Chai!!.
The code in Step 4 uses a LINQ query to get all products from the Northwind database and bind the data to the GridView. The code in Step 5 also uses a LINQ query, but it gets a subset of the products — just the products that start with the letter C. `StartsWith()` is a built-in function. The `For Each` loop acts on the subset of products and alters the product names by calling the `ToExclaim()` extension function that you create at the start of this section. IntelliSense offers the `ToExclaim()` function just like for the big boys (refer to Figure 8-6).

After changing the names of products that start with C, the code calls the `SubmitChanges()` method to make the changes happen in the database.

If you’re using LINQ to SQL to insert or update data, you must call `SubmitChanges()` to push the content into the database.

If you don’t like exclamation marks cluttering your product names, change the `For Each` loop in the code of Step 5 to use the following and click again:

```csharp
prod.ProductName = Replace(prod.ProductName, "!", "")
```

## Inserting Data with the DataContext

You don’t use LINQ to SQL to insert data because you’re not acting on a set of data. Instead, you use the `DataContext` classes that you create earlier in the section, “Creating the database access code.” Take a look at the code in Step 3; the key is the `DataContext` object’s `InsertOnSubmit()` method that acts like `Add()` in most collections. Follow these steps to create a new `Product` object, configure it, and add it to the database:

1. **Use the exclaim.aspx page you created in the preceding section, “Updating Data with a LINQ Query.”**
2. **In Design view, add a Button control with the text** `Insert` **and double-click the button to create a handler for the Click event.**
3. **In Source view, add the following code to the handler subroutine:**

```csharp
Dim dc As New NWDataClassesDataContext
Dim prdcts = dc.Products
Dim prod = New Product
prod.ProductName = "CoolJuice"
prod.QuantityPerUnit = "8 per box"
prod.UnitPrice = 5
prod.SupplierID = 2
prod.CategoryID = 1
prod.UnitsInStock = 5
prod.ReorderLevel = 1
prdcts.InsertOnSubmit(prod)
dc.SubmitChanges()
BindData()
```
4. At runtime, click the Insert button that you add in Step 2.

The preceding code adds CoolJuice as the last item in the list of products.

The code in Step 3 creates a DataContext object that represents the database tables and uses the variable `prdcts` to hold a reference to the Products collection. The variable `prod` represents a Product object, created from a class generated by the object relational designer.

With the new `prod` object, you configure associated properties, such as `ProductName` and `ReorderLevel`. After the product is configured, add the product to the Products collection (represented by `prdcts`) using the `InsertOnSubmit()` method. Note that you still need to submit the new product object to the database by calling `SubmitChanges()`. The `BindData()` method ensures that the DataGridView refreshes to display the latest version of the data.
Chapter 9

Creating and Consuming Diverse Data

In This Chapter

- Putting syndicated RSS feeds on a page
- Using the XmlDataSource and DataList controls
- Adding style to XML data
- Shaking the daylight out of TimeZoneInfo
- Creating Web and WCF services

Web-based data doesn’t just come from Microsoft SQL Server. Data exists as blog feeds in RSS, as bits and bytes in Access databases, as XML markup, and even embedded in HTML tags.

One attraction of XML is that you can massage it to suit your needs, as you see in this chapter. What’s more, by delivering XML content with a Web service, you go a long way to breaking down data silos and incompatibilities.

Putting an RSS Feed on a Page

RSS is a format for an XML document that describes a list of items, such as blog entries or news headlines. The document may just be a stream of XML created on the fly rather than a static file. In this section, you put an RSS document (using Version 2.0) on a page by using the XmlDataSource and DataList controls.

Analyzing an RSS feed

RSS Version 2.0 presents a hierarchy of nodes. A stripped-down RSS structure looks like this:
To retrieve text within a `<title>` element, you need to stroll through `<rss>`, `<channel>`, and `<item>`. In XML, you can describe the route to the `<title>` text by using XPath syntax. As you see in the next section, the ASP.NET XmlDataSource control does the strolling for you.

### Using the XmlDataSource control

The XmlDataSource control has an XPath property that bores into an XML document to make finding the data you want easy. Follow these steps to point an XmlDataSource to an RSS feed:

1. **From the Toolbox, add an XmlDataSource control to an ASP.NET page.**
2. **From the XmlDataSource Tasks menu (click the Smart Tag button), select Configure Data Source.**
3. **In the Data file text box, enter the URL for the RSS feed.**
   
   For example:
   
   ```
   http://www.microsoft.com/presspass/rss/TopStory.xml
   ```
   
   The URL may use extensions such as .aspx, .php, and so on. What’s important is that the location produces an RSS XML document.
4. **In the XPath expression text box, enter the following expression and then click OK:**
   
   ```
   rss/channel/item
   ```
   
   This line tells the control that you want to start fetching data this far into the structure of the document.

The XmlDataSource control returns data but doesn't display it. You can use many controls to show the results, including the ASP.NET DataList.
Displaying XML data by using the DataList

The preceding section told the XmlDataSource roughly where to get the data. Using the DataList control, you get to the specific fields you want to display. Follow these steps to use an ASP.NET HyperLink control to render the titles from the RSS feed and link to the content.

1. Add a DataList control to the ASP.NET page.
   Setting the DataList control’s EnableViewState property to False reduces the number of bytes that need to be transferred at runtime.

2. From the Tasks menu (click the Smart Tag to open it), select XmlDataSource1 (or whatever yours is called) from the drop-down list.

3. Click the Edit Templates link and display the ItemTemplate template.

4. From the Toolbox, drag a HyperLink control and drop it inside the ItemTemplate area.

5. From the HyperLink control’s Tasks menu, select Edit DataBindings.
   The HyperLink1 DataBindings window opens with a list of bindable properties.

6. In the Bindable Properties area, select NavigateUrl; in the Custom binding box, enter the code expression XPath("link").

7. Select the Text bindable property; in the Custom Binding box, enter the code expression XPath("title") and click OK.

Steps 6 and 7 in the preceding list tell the HyperLink control to display the content from the title node of the RSS document but use the contents of the link node as the destination URL.

Browse to the page to view the RSS content as hyperlinked text.

The DataList is a templated control, which means you can redesign its look with your own markup and styles (or use the Autoformat feature). For more on templated controls, see Chapter 13.
Making an RSS Feed Available from Your Site

You can create an RSS feed by sending any list of data as an XML stream. The LINQ to XML capabilities in ASP.NET 3.5 make generating XML on the fly easy. Follow these steps to create an ASP.NET handler that produces an RSS feed:

1. Add a generic handler named `rsshandler.ashx` to your project (File ➤ New File ➤ Generic Handler ➤ Add).
2. Add the statement `Imports System.Xml.Linq` below `Imports System`.
3. Replace the existing `ProcessRequest()` subroutine with the contents of Listing 9-1.

When you browse to `rsshandler.ashx` with Internet Explorer 7, the browser recognizes that it's RSS. IE 7 formats the content and makes it easy to subscribe to the feed (see Figure 9-1).

Listing 9-1: An RSS Feed Using LINQ to XML

```csharp
Public Sub ProcessRequest ByVal context As HttpContext Implements IHttpHandler.ProcessRequest
    Dim rssdoc As New XDocument(New XDeclaration("1.0", Nothing, Nothing)) ➔4
    rssdoc.Add(New XComment("XML to LINQ Generated RSS Feed")) ➔5
    Dim rssrootelement As New XElement("rss", New XAttribute("version", "2.0")) ➔6
    Dim rsschannel As New XElement("channel")
```
Here’s what you need to know about Listing 9-1:

→4-5 RSS, like all XML, resembles those Russian nesting dolls. The XDocument object is the largest container “doll” and includes “decorations,” such as XDeclaration and XComment.

→6-12 The rssrootelement variable holds the root node, the <rss> tag. The code generates a <channel> element that it adds to the root. Likewise, it adds <title>, <description>, and <link> elements inside the <channel>.

→17-25 When you reach the <item> elements, the code uses a For...Next loop to create sets of <title>, <description>, and <link> elements that it adds to <item> elements on each pass through the loop. The <item> elements nest inside a <channel>.

→28 Finally, the whole document is ready, and you use the XDocument object’s Save() method to save the contents as a StreamWriter object that finally goes out to the browser as HTML.

To understand what’s happening in Listing 9-1, it helps to glance at the generated RSS feed as abbreviated in Listing 9-2.
Listing 9-2: Excerpt of Generated RSS Feed

<?xml version="1.0" encoding="utf-8"?>
<!--XML to LINQ Generated RSS Feed-->
<rss version="2.0">
  <channel>
    <title>Ken's Sample Channel</title>
    <description>Description of Channel</description>
    <link>http://www.kencox.ca/</link>
    <item>
      <title>This is item number 0</title>
      <description>Description for item # 0</description>
      <link>http://www.kencox.ca/item0.aspx</link>
    </item>
    <item>
      <title>This is item number 1</title>
      <description>Description for item # 1</description>
      <link>http://www.kencox.ca/item1.aspx</link>
    </item>
  </channel>
</rss>

Transforming XML Data into HTML Markup

With XML, if you don’t like the existing tags or structure, you can change it into whatever format suits you or your application. An Extensible Stylesheet Language (XSL) transformation pulls data from XML nodes and renders the content according to your instructions. In this example (shown in Figure 9-2), you start with an XML file containing details about Web sites. You render the raw data as attractive HTML on an ASP.NET page.
Gathering the source XML data

The raw material for a transformation is XML data. Follow these steps to create the XML file:

1. Add a new XML file named aspsites.xml to the App_Data folder of your project (File ➤ New File ➤ XML File ➤ Add).
2. Use Listing 9-3 as the complete contents of aspsites.xml.

The source file has a simple hierarchical structure: The root, sites, contains one or more site nodes. Each site has a title, url and description node.

Listing 9-3: Sample Raw Data as XML

```xml
<?xml version="1.0" encoding="utf-8" ?>
<sites>
  <site>
    <title>The Official Microsoft ASP.NET Site</title>
    <url>http://www.asp.net/</url>
    <description>The primary site for ASP.NET developers.</description>
  </site>
  <site>
    <title>4 Guys From Rolla</title>
    <url>http://aspnet.4guysfromrolla.com/</url>
    <description>An amazing collection of free tutorials.</description>
  </site>
</sites>
```

Creating the XSL style sheet

XSL is a style sheet language that can transform XML markup into a completely different format. In this example, you convert the source document into HTML markup and render it in a browser. Follow these steps to create the XSL style sheet.

1. Add a new XSLT file named xsltfile1.xsl to the root of your project (File ➤ New File ➤ XSLT File ➤ Add).
   The default XSLT file assumes that you want to transform XML into a complete XHTML page, but that’s not the case in this example.
2. Replace the existing markup in xsltfile1.xsl with the contents of Listing 9-4.
Here’s what you need to know about Listing 9-4:

5. The working part of the XSLT file starts with `<xsl:template match="/sites">`, which indicates that it should deal with all nodes within the `sites` node.

6. The code starts a `for each` loop (`<xsl:for-each select="site">`) that acts on every `site` node — the sample XML file has only two `site` nodes.

7. The transformation inserts HTML, such as the `<p>` tag. The statement `<xsl:value-of select="title" />` extracts the content of the `<title>` tag and, at runtime, substitutes the value.

8. More HTML appears, along with another extraction, `<xsl:value-of select="url" />`, to insert the URL of the site.

13. The closing tag of the `for each` loop tells the transformation to start over with the next `site` node (if any nodes are left, that is).

**Using the ASP.NET Xml control**

ASP.NET’s `Xml` control manages the technical details of applying a transformation to an XML file and putting the results on a page. Follow these steps to use the `Xml` control to perform an XSL transformation with the preceding style sheet and data files:

1. **Add an ASP.NET page named** `transform.aspx` **to your project.**

2. **From the Toolbox, add an ASP.NET** `Xml` **control to your page.**

3. **Using the control’s Properties window (F4), set the** `DocumentSource` **property to the location of the XML data file,** `App_Data/aspsites.xml`. 
4. Set the TransformSource property to the location of XSLTFile1.xsl.

5. Change the EnableViewState property to false.
   If you don’t need to track changes or updates to data, you can reduce the overall page size by disabling the ASP.NET ViewState.

   ASP.NET carries out the transformation so that the browser sees only the HTML markup and the data values.

Connecting Web Applications to an Access Database

For busy ASP.NET Web applications, Microsoft SQL Server is probably your best choice as a database. However, you don’t always get the choice. For example, someone may hand you an Access database and ask you to build a Web page to display and update the contents.

This example uses the Access version of the Northwind database. If you don’t already have a copy of nwind.mdb on your computer, you can download a copy from the Downloads area of Microsoft’s Web site by searching for Northwind Traders Sample Database. Follow these steps to install and connect to an Access database:

1. In your ASP.NET project, copy nwind.mdb into the App_Data folder.
3. In Design view, from the Toolbox, drag an AccessDataSource control and drop it on the page.
4. Using the AccessDataSource Smart Tag, open the Tasks menu and select Configure Data Source.
5. In the Configure Data Source Wizard, browse to the Access data file in the App_Data folder, click OK, and then click Next.
6. Select the table name (for example, Customers) and check the asterisk (*) for all columns.
7. Click the Advanced button, check Generate INSERT, UPDATE, and DELETE statements, and then click OK.
8. Click Next and then click Finished.
After you configure an AccessDataSource control, it takes only a minute to configure a control to view and update the data. The ListView control connects easily with an AccessDataSource control. Follow these steps to manipulate the data with a ListView control:

1. From the Toolbox, drag and drop a ListView control on the page.
2. From the ListView’s Tasks menu, click the drop-down list and select the AccessDataSource control’s ID.
3. On the Tasks menu, select Configure ListView.
4. In the Configure ListView window, select the Grid layout, Professional style, and all the options (editing, inserting, and so on) and then click OK.
5. Run the page and edit, update, and delete the records as necessary.

If you get an error message, such as OleDbException (0x80004005): Unspecified error, it usually means that ASPNET, Network Service, or the current user’s account doesn’t have Modify or Write permissions in the special App_Data folder. Those permissions are necessary to let Access create its temporary files. For more information on the necessary permissions to run ASP.NET applications, search for ASP.NET Required Access Control Lists on the Microsoft Web site.

Creating a Simple Web Service

Web services are convenient ways of making functions available across the Internet or intranet. They’re part of the “programmable Web” because programs and Web pages running on any platform can submit data and get results.

The simple Web service you create in this section calculates the red, blue, and green values for any of the “known” colors in .NET. For example, a program submits the word “pink” and the Web service returns a string containing values, such as “255:192:203” (meaning “red:green:blue”). It’s up to the consumer program to parse and use the results.

To create the GetRGB Web service, follow these steps:

1. Add a Web service file named rgbservice.asmx to your project (File ‹ New File ‹ Web Service ‹ Add).
2. Use Listing 9-5 as the complete contents of the Web service.
3. Browse to rgbservice.asmx.

   ASP.NET’s built-in documentation page appears.
4. Click the GetRGB link to reach the test page.

5. Type *green* in the text box and click Invoke (see Figure 9-3).

   The Web service responds with the color’s value wrapped in an XML message:

   ```xml
   <?xml version="1.0" encoding="utf-8"?>
   <string xmlns="http://kencox.ca/">0:128:0</string>
   ```

   **Figure 9-3:** The ASP.NET Web service test page.

---

**Listing 9-5: GetRGB Web Service**

```xml
<%@ WebService Language="VB" Class="rgbservice" %>
Imports System.Web
Imports System.Web.Services
<WebService(Description:="Returns RGB values for a known color", _
   Name:="Ken's RGB Service", Namespace:="http://kencox.ca/")> _
<WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
Public Class rgbservice
   <WebMethod()> _
   Public Function GetRGB (ByVal strKnownColor As String) As String
      Dim colr As System.Drawing.Color
      Dim rgx As New Regex("^[a-zA-Z]{1,30}$")
      If Not rgx.IsMatch(strKnownColor) Then
         Return "Bad input"
      End If
      colr = System.Drawing.Color.FromName(strKnownColor)
      Return colr
   End Function
End Class
```

(continued)
Listing 9-5 (continued)

If (colr.R = 0) And (colr.G = 0) And (colr.B = 0) And (strKnownColor.ToUpper <> "BLACK") Then
    Return 'Not known'
Else
End If
End Function
End Class

Listing 9-5 includes plumbing to make Web services work in ASP.NET.

13-19 A key attribute is <WebMethod()> (a *decoration* in geekspeak) that exposes the GetRGB() function as code that outsiders can use. The function uses a regular expression to block out bad input by declaring that only the letters *a* to *z* are allowed, and there must be at least one, but not more than 30 characters. Anything that doesn’t fit that pattern merits a *Bad input* message.

21-23 This section converts the color name, such as pink, into a Color object. A Color object has distinct red (R), green (G), and blue (B) properties that you can test. If you try to convert an unknown color, all the return values are zero — the same as for Black. So, the code makes sure that it didn’t get a legitimate request for Black.

24 If the color name isn’t one that .NET knows about, the function returns the string *Not known*. If it’s a usable color, the final statement concatenates the values (separated by colons) and returns that string.

You can use a Web service to return calculations or data. The consumer — who can be in the same building or on the other side of the world — only needs to know what parameters your Web service requires. The implementation details aren’t important.

ASP.NET hides the test page shown in Figure 9-3 when you view the service from the Internet. To override this security precaution, put these elements in the *web.config* file after the <system.web> element:

```xml
<webServices>
    <protocols>
        <add name="Documentation"/>
        <add name="HttpGet"/>
        <add name="HttpPost"/>
    </protocols>
</webServices>
```
Adding a Web Reference to a Project

You can connect to the preceding Web service by creating a Web reference. You then consume the service in an ASP.NET Web page.

Before a page can use functions on another server, it needs to know where to find the service and how to invoke the functions. Visual Web Developer includes graphical tools for discovering a Web service and creating the connection. Follow these steps to add a Web reference:

1. **In Solution Explorer, right-click the project name and choose Add Web Reference from the context menu.**

2. **In the URL box, enter the Internet address of the Web service — for example, http://www.kjopc.com/rgbservice.asmx — and then click Go.**

   If your Web service is in the same project, click the Web Services In This Solution link (in the lower pane) to find it.

   As shown in Figure 9-4, the dialog box connects to the URL and reports on the Web services that it finds.

3. **In the Web Reference Name box, type the name of this reference (for example, getrgbservice) and then click Add Reference.**

   The IDE puts the Web reference into a new folder called App_WebReferences in Solution Explorer. It also adds an entry to the web.config file that includes the URL and name of the Web service. On my system, it looks like the following:
Keep the URL current. For example, during development, your Web service URL may be http://localhost/myproject/rgbservice.asmx. However, when you deploy the Web service to the Internet, you must revise the address in the web.config to something like http://www.kjopc.com/rgbservice.asmx.

Creating a Page to Use the Web Service

Follow these steps to create a page that uses the GetRGB() function from the preceding Web service.

1. Add an AJAX Web form (yes, there’s an AJAX version!) named checkcolor.aspx to your project (File ➪ New File ➪ AJAX Web Form ➪ Add).

2. In Design view, from the AJAX Extensions category of the Toolbox, drop an UpdatePanel control onto the page.

   Make sure that the ScriptManager control remains at the top of the page, or you get a runtime error.

3. Place your cursor inside the UpdatePanel and then from the Toolbox, double-click an ASP.NET TextBox, a Button, and a Label control.

   It’s hard to place controls inside other controls. The double-click technique ensures that the controls land inside the UpdatePanel.

4. In Design view, add an UpdateProgress control below the UpdatePanel control.

5. Place your cursor inside the UpdateProgress control and, from the Toolbox, double-click a Label control.

6. Change the preceding Label control’s Text property to An update is in progress....

7. Double-click the Button control to create a default handler subroutine for its Click event and add the following code within the subroutine:

   ```vbscript
   Dim ws As New getrgbservice.KensRGBService
   Dim strColor As String
   strColor = TextBox1.Text
   Label1.Text = ws.GetRGB(strColor)
   ```
The preceding code uses the variable `ws` to create an instance of the Web service. It fetches the color name from the text box and passes the color name to the `ws.GetRGB()` function. The return value appears in the `Label` control as its `Text` property value.

If you get an error after deploying the page and Web service, check that you copied the `App_WebReferences` folder and its content and that the URL of the Web service is correct in the `web.config` file. For more on deployment, see Chapter 22.

To test the page and the Web service, browse to `checkcolor.aspx`, type `silver` in the text box and click the button. If the connection is slow, you see the `UpdateProgress` control reporting progress. After that, the Web service returns the result string `192:192:192`, which means that the color silver in .NET uses 192 for each of the red, green, and blue values.

The Visual Basic `Split()` function is handy for parsing the numbers out of the returned string and into an array. You need to tell it the string to parse and the character that separates (delimits in geekspeak) the elements. Here’s the idea:

```vbnet
Dim arr As Array
arr = Split(ws.GetRGB(strColor), ":")
Label1.Text = (arr(0).ToString)
```

To simulate a four-second delay for testing the `UpdateProgress` control, add this line of code to your Web service `GetRGB()` function:

```vbnet
Threading.Thread.Sleep(4000)
```

---

**Creating a Daylight Saving WCF Service**

Windows Communication Foundation (WCF) is Microsoft’s latest technology for making functions and data available across networks like the Internet. In WCF, you define a `contract` that describes what your service does and what it expects as input. There’s also a `binding` requirement that defines the transport method(s) to use, such as HTTP used in these examples. The final element is the `endpoint`, or address, where users can find the service.

This WCF service accepts the name of a time zone (such as eastern standard time) and returns details about when daylight savings time is in effect. Follow these steps to implement the `GetTimeZoneInfo` service:
1. In Visual Web Developer, create a WCF Service site named time service (File ➪ New Web Site, choose WCF Service and click OK).

VWD generates several files for you, including Service.svc and Web.config in the project’s root, and IService.vb and Service.vb within the App_Code folder.

2. Open IService.vb in the App_Code folder and insert the following code immediately below the TODO comment and above End Interface.

   <OperationContract()>
   Function GetTimeZoneInfo(ByVal strLocalTimeZone As String) As String

   This code defines a contract, the GetTimeZoneInfo() function.

3. Open Service.vb in the App_Code folder and add the contents of Listing 9-6 just above the existing End Class statement.

4. Open the web.config file and in the <system.serviceModel> element, locate the first <endpoint> element, and change the binding value from wsHttpBinding to basicHttpBinding as shown here:

   <endpoint address="" binding="basicHttpBinding"
   contract="IService"/>

You can browse to Service.svc, but the documentation page isn’t terribly helpful because it’s intended for rich clients (like Windows forms applications) rather than for use in a Web form.

Listing 9-6: The GetTimeZoneInfo and Formatting Functions

```vbnet
Public Function GetTimeZoneInfo(ByVal strLocalTimeZone As String) As String Implements IService.GetTimeZoneInfo
    Dim sb As New StringBuilder
    Dim tst As TimeZoneInfo
    Try
        tst = TimeZoneInfo.FindSystemTimeZoneById(strLocalTimeZone)
        Catch exc As TimeZoneNotFoundException
            Return "Can't find the time zone " & strLocalTimeZone & "."
        Catch exc As Exception
            Return exc.Message
        End Try
    Dim tzadj As TimeZoneInfo.AdjustmentRule
    For Each tzadj In tst.GetAdjustmentRules()
    Next
    Return sb.ToString
```
Here’s how Listing 9-6 breaks down:

1-12 The `GetTimeZoneInfo()` function accepts a time zone name as a parameter and uses the name in the `FindSystemTimeZoneById()` method to return a `TimeZoneInfo` object. The `Try...Catch...End Try` sequence handles the case where the supplied time zone can’t be found.

13-14 Inside the `TimeZoneInfo` object, you find an `AdjustmentRule` object that holds rules for time shifts, such as starting and ending daylight savings time. The `AdjustmentRule`’s `GetAdjustmentRules()` method returns a collection of rules. The `For Each` loop examines each rule for its details about the transition into and out of daylight time.

24 The `CTT()` function’s role is to dig out information from the adjustment rule and put the values into an English sentence. For example, instead of referring to the 1 week of the month, people use an adjective, such as first.

31-34 In the same vein, the Visual Basic `WeekDayName()` function converts a day number into English text, such as Saturday, and `MonthName()` turns month numbers into names, such as March. The routine uses the `StringBuilder`’s `Append()` method to assemble the text that it returns as a string to the calling function.

### Creating the Service Consumer Web Form

In this section, you build a Web page that uses the preceding WCF service. As with a Web service, Visual Web Developer handles most of the plumbing to discover what the service expects as parameters and what it returns.
The Web form in this example displays a list of time zones that uses daylight time for part of the year. The user selects a time zone and clicks a button. This action transmits the time zone name to the WCF service and displays the calculated results. To create the consumer page, follow these steps:

1. **Add an ASP.NET page named usesvc.aspx to the project.**
   This book uses the single-file model for ASP.NET pages, so you need to uncheck the Place Code In Separate File check box.

2. **In Design view, add DropDownList, Button, and Label controls to the page.**

3. **Double-click a blank area of the page to create a default handler routine for the Page Load event and then add the following code inside the routine:**

   ```vbnet
   If Not IsPostBack Then
   Dim q = From tz In _
       TimeZoneInfo.GetSystemTimeZones() _
       Where tz.SupportsDaylightSavingTime _
       Select sname = tz.StandardName, _
       dname = tz.DaylightName Order By sname
   DropDownList1.DataSource = q
   DropDownList1.DataTextField = "dname"
   DropDownList1.DataValueField = "sname"
   DropDownList1.DataBind()
   End If
   ```

   This code uses a LINQ query to look through the Web server’s time zone information and select only the time zones that support daylight time. The query extracts the standard time zone name and the daylight time name and then tells the drop-down list to use that information as its data.

4. **In Design view, double-click the Button control to create a default handler for the Click event.**
   You insert the code into the handler in the next procedure.

You can run the page to see the drop-down list containing the names of daylight time zones sorted alphabetically. The next step is to figure out where to send and receive the data.

**Connecting to a WCF Endpoint**

When VWD creates a Windows Communication Foundation service, it makes the service discoverable by clients. You can see the Web Service Description Language (shortened to WSDL and pronounced *wizz-dell* in geekspeak) details by tacking on a query string to the URL:

```
http://www.kjopc.com/service.svc?wsdl
```
Everything you need to know about connecting to the service is in the WSDL definition. Fortunately, the built-in tools decipher it for you and create whatever else you need to connect.

Follow these steps to connect to the WCF service:

1. Choose Website ➪ Add Service Reference.
   The Add Service Reference window appears.

2. In the Address text box, enter the URL of the daylight saving WCF service (for example, http://www.kjopc.com/service.svc) and click Go.
   The utility downloads the service information (this step can be slow) and lists the available services and operations, as shown in Figure 9-5.

3. Accept the default values in the Add Service Reference window and click OK.
   VWD puts several files into a new folder called App_WebReferences.

4. Open the web.config file and locate the <system.serviceModel> element and within it, the <client> element.

5. In the <endpoint> element, check that the address value is pointing to the URL where the service is located and fix it, if necessary.

For example, the endpoint address for my service is

```xml
<endpoint address="http://www.kjopc.com/service.svc"
```

Figure 9-5: Adding a WCF service reference.
The Add Service Reference utility sometimes has a mind of its own about the address value. It may ignore the IP address or URL that you gave it in Step 2, which is why you need to check it.

6. **In Source view of usesvc.aspx, inside the Button1_Click handler routine, add the following code:**

```vbnet
Dim srvc As New ServiceReference1.ServiceClient
Dim tzone As String = DropDownList1.SelectedValue
Label1.Text = Replace(srvc.GetTimeZoneInfo(tzone), _
    Environment.NewLine, "<br />")
```

The preceding code creates an object from the WCF service. Then it gets the selected time zone name from the drop-down list. The last line passes the time zone name to the service’s `GetTimeZoneInfo()` function. The string that comes back from the WCF service includes line breaks that HTML doesn’t recognize, so the code uses the Visual Basic `Replace()` function to put in an HTML line break (`<br />`) wherever it finds a `Environment.NewLine` value. Finally, the routine tells `Label1` to display the string as its `Text` property.

Although ideally you specify the URL of the WCF service in the `web.config` file, you can also hardcode it when you create the service object:

```vbnet
Dim srvc As New ServiceReference1.ServiceClient _
    ("BasicHttpBinding_IService", _
    "http://www.kjopc.com/service.svc")
```

Run the page, select a time zone, and click the button. The WCF service returns the daylight time information, as shown in Figure 9-6.
Part III

Enhancing the Interface and User Experience

The 5th Wave  By Rich Tennant

“I have to say I’m really impressed with the interactivity on this car wash Web site.”
In this part . . .

This part focuses on creating functional, attractive, and nimble Web pages while promoting labor-saving techniques and reusing code. In Chapter 10, you see how a master page, style sheet, or skin controls the appearance of every ASP.NET page in a Web site. In Chapter 11, you step through the navigation controls to ensure that visitors can find your great content. Chapter 12 deals with HTML standards, page layout, and usability. Chapter 13 concentrates on the ListView control, which is introduced in ASP.NET 3.5. ListView grants you total control over its layout while the control worries about the data-handling details.

In the second half of Part III, I introduce you to visual effects, which add polish to your pages. In Chapter 14, you use styles to create rollover links that look like buttons. You also generate custom images for each visitor. Chapter 15 takes you into the free AJAX Control Toolkit where I show you how to spice up the basic ASP.NET controls at no cost — and almost no effort. The final chapter of this part covers Silverlight and Flash content.
Creating a top-notch UX (User eXperience) requires extra effort and attention to the appearance of Web pages. A professional design brightens even the most mundane text. You see this effect with grids, charts, and graphs; those gently rounded, blended, shadowed, and sculptured chart objects look almost, well, edible.

Unfortunately, this chapter won’t turn you into a page designer. However, it does show you how to create the canvas for your design and style that you can apply to ASP.NET controls. You also see how to create templates for Web applications, where you mix and match skins, themes, and master pages.

Deciding Where Style Rules Belong

You wouldn’t think that creating a blue button on a Web page could involve much thought and introspection. “Just color it blue and move on to the next one!” you insist.

The trouble is you can store styles in several places. Choosing the wrong spot can add to your workload. Worse yet, you might end up with styles scattered all over the place with some overriding the effect of others.
Quick and not-too-dirty with AutoFormat

Using Visual Web Developer, you can pretty-up controls, such as the GridView control, with a few mouse clicks. Select the control, open its Smart Tasks, click the AutoFormat link, and you’ve got style.

The AutoFormat window for GridView, shown in Figure 10-1, includes built-in styles, such as Colorful, Professional, and Black & Blue. You can apply one of the AutoFormat styles as a starting point and then tweak the font, colors, spacing, and other properties to suit your needs. (Just be careful not to overwrite your changes by reapplying the original AutoFormat.)

Your choice of this method indicates that you need instant gratification. You’re not too concerned with any long-term, site-wide consequences of embedding style information directly into the control’s markup.

Keeping styles close and inline

You can select a GridView (or any visual control), open its Properties window (F4), and add all sorts of style information. The Appearance category with values for the BackColor, BorderStyle, and ForeColor properties.

When you apply styles by using the Properties window, you’re inserting the style information directly into the control’s markup. The effect at runtime is an inline style, so called because the details, such as the font, color, and border width, are jammed into the line that describes the control. To geeks, the following style markup hardcodes values, which is something to avoid:
Inline styles create a maintenance problem as your site grows. Say you decide that the red border on the grids is too garish. On a large site, you'd need to open every page, find every control, locate the property, and change its value. The next section discusses separating the style from the control.

**Storing styles in the page’s `<style>` tag**

Style embedding moves the style information away from the control — but keeps it within the same page. To try it out, add the following markup within the HTML `<head>` section:

```html
<style type="text/css">
.gridstyle1
{
  background-color: #cccccc; border-color: red;
  border-style: solid; border-width: 3px;
  color: Black; font-weight: bold;
}
</style>
```

Geeks refer to `gridstyle1` as a style class. Having separated the style from the GridView, you must tell the grid control where to look for its style information. That’s done with the `CssClass` attribute, as shown in the following:

```html
<asp:GridView ID="GridView1" runat="server"
CssClass="gridstyle1" AutoGenerateColumns="False"
CellPadding="4" CellSpacing="2" DataKeyNames="ProductID"
DataSourceID="SqlDataSource1" />
```

**Storing styles in an external CSS style sheet**

To a `<head>` shrinker (that’s a little HTML pun), using an external style sheet shows you value reusable elements and efficiency. Here’s how to add an external style sheet file and style class to your project and pages:
1. In Solution Explorer, add a style sheet to the project (Add New Item ➪ Style Sheet).

2. Add the following style class markup to the style sheet:

```css
.gridstyle1
{
  background-color: #cccccc; border-color: red;
  border-style: solid; border-width: 3px;
  color: Black; font-weight: bold;
}
```

3. Open the ASP.NET page in Design view.

4. From Solution Explorer, drag the style sheet’s filename (for example, StyleSheet.css) and drop it on the page.

   An external link is created inside the `<head>` section.

5. Select the ASP.NET control and open its Properties window (F4).

6. Set the `CssClass` property to `gridstyle1`.

Every page in your site can link to the same style sheet file and make the `gridstyle1` style class available to its contained controls.

**Using the VWD Style Sheet Tools**

Creating styles is a visual task in which real *artistes* don a beret and painter’s smock, choose a style, such as a border color, apply it to a control, and then stand back to admire their work. In this section, you use the graphical tools in Visual Web Developer to design and preview CSS styles.

**Attaching an external style sheet**

In preparation for designing styles, you need a place to store your precious artistry. Follow these steps to attach a style sheet file to a page:

1. In Solution Explorer, add a style sheet file to the project (Add New Item ➪ Style Sheet).

2. Add an ASP.NET page to your project (for example, `ssheet.aspx`) and open it in Design view.

3. Open the Apply Styles window (View ➪ Apply Styles).

   Figure 10-2 shows the Apply Styles window.

4. Click Attach Style Sheet.

   The Select Style Sheet dialog box opens.
5. Navigate the project folders to find the style sheet to attach and then click OK.

The style sheet filename (for example, StyleSheet.css) appears in the Apply Styles window.

From here, you need a control and a style. That’s coming right up!

To “unattach” the attached style sheet, right-click the name of the style sheet in the Apply Styles window and choose the Remove Link item from the context menu.

### Adding a style rule to an external style sheet

At this point, you have a style sheet but nothing useful inside it. In this section, you create a new style rule that’s destined for use on one or more ASP.NET TextBox controls. Follow these steps to add a style rule:

1. Open the Apply Styles window (View ➪ Apply Styles), and click the New Style link.

   The New Style dialog box appears.

2. In the Selector box, replace the default name with .wildTextBox, as shown in Figure 10-3.

   The dot (.) prefix is significant because it indicates that this style rule applies to all controls that reference it by using cssclass= "wildTextBox".

3. In the Define In box, from the drop-down list, choose Existing Style Sheet.
4. In the URL box, select the name of the style sheet (for example StyleSheet.css). If the filename isn’t on the list, click Browse to find it.

5. Click OK.

The dialog box closes and the Apply Styles window reappears. The style name (.wildTextBox) appears in the list of available styles.

**Splashing on some wild style**

For visual impact, I’m showing you how to go a bit crazy with the .wildTextBox style. After seeing it, you may decide against adopting my design as your corporate standard. However, if you like it, use it and please send me the URL! To configure a style, follow these steps:

1. **Open the Apply Styles window (View ➪ Apply Styles)**
2. **Right-click the existing style (.wildTextBox) and from the context menu, choose Modify Style.**

   The Modify Style window opens.

3. **In the Category box (it’s on the left side) select the Font category and type or choose the property values, as shown in Figure 10-4 and in the following table:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>font-family</td>
<td>Comic Sans MS</td>
</tr>
<tr>
<td>font-size</td>
<td>large</td>
</tr>
<tr>
<td>font-weight</td>
<td>bold</td>
</tr>
<tr>
<td>color</td>
<td>blue</td>
</tr>
</tbody>
</table>

4. **In the Category box, select the Background category and set the background-color property to yellow.**
5. In the Category box, select the Border category and then set the border properties shown in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>border-style</td>
<td>dotted</td>
</tr>
<tr>
<td>border-width</td>
<td>3px</td>
</tr>
<tr>
<td>border-color</td>
<td>red</td>
</tr>
</tbody>
</table>

6. In the Category box, select the Background category and then set the background-color property to yellow.

7. Click OK to save the styles.

As shown in Figure 10-5, the name of the style (.wildTextBox) takes on the wild style attributes so you can see what you’re getting. Nice touch!

---

**Applying a style to a TextBox control**

Your outlandish new corporate style is ready for its public debut. All you need is a skinny TextBox control to model your Pierre Cardin original on the virtual runway. Here’s how to apply the style:

1. **In Design view, add an ASP.NET TextBox control to your page.**
2. Open the Apply Styles window (View ➪ Apply Styles).
3. Select the TextBox control.
4. In the Apply Styles window, click the style name (.wildTextBox).

The text box comes alive by adopting the bizarre style. When you run the page and enter some text, you see blue letters with a charming dotted red border.

Be careful about what you select on the ASP.NET page when you have the Apply Styles window open. You can easily apply an unwanted style to the selected control or even the page. If it happens to you, delete the CssClass value from the control.

**Analyzing the generated style**

While you were having fun with the wild style, VWD was following good practices in the use of style sheets. If you view the ASP.NET page in Source view, you don't find any style markup. For example, the text box markup contains only a reference to the style, as shown in bold:

```html
<asp:TextBox ID="TextBox1" runat="server"
CssClass="wildTextBox"></asp:TextBox>
```

VWD inserted the formatting details into the external style sheet, StyleSheet.css:

```
.wildTextBox {
    border: 3px dotted #FF0000; color: blue;
    font-family: "Comic Sans MS";
    background-color: yellow;
    font-size: large; font-weight: bold;
}
```

**Managing Style Rules**

You see that style rules can live in many places: inline next to the control; embedded within the page; and stored in an external style sheet. VWD helps you track all these style rules and even shift them from one location to another.
Moving styles from a page to a style sheet

Here’s a typical situation: You’re adding to the Web site and realize that you’ve embedded styles in several pages. You want to consolidate the embedded styles into one external style sheet. Rather than embark on a frantic cut-and-paste blitz, follow these steps to move embedded styles to an external style sheet:

1. If your project doesn’t have an external cascading style sheet, add one (File ➪ New File ➪ Style Sheet).

2. Open an ASP.NET page in Design view that includes embedded style rules or insert a style such as this:

   ```html
   <style type="text/css">
   .rulez
   {font-size: xx-large;}
   </style>
   ```

3. Open the Manage Styles pane (View ➪ Manage Styles), as shown in Figure 10-6.

4. In the Manage Styles pane, from the Current Page category, drag a style name and drop it on the name of the external style sheet (StyleSheet.css in this example).

   You can confirm that the style was moved by checking the ASP.NET page in Source view.

   You can move several styles by selecting the first in the list and Shift-click the last in the list. The selected styles move as one.
Adding, modifying, and deleting styles

Use the Manage Styles pane to add, change, and delete styles. To open the Manage Styles pane, choose View ▸ Manage Styles.

You can delete a style by selecting the style name and pressing the Delete key. To add a style, click the New Style link. To modify a style, select the style name, right-click, and select Modify Style from the context menu.

Using Master Pages with Slavish Devotion

Chances are, most of the Web pages in your site have something in common. For example, they might use the same banner, menu, login link, background pattern, and copyright blurb. Master pages in ASP.NET let you collect those common chunks in one central file instead of duplicating the code in every page.

Master pages include placeholder areas where you plug in your regular, changeable page content. At runtime, ASP.NET merges all the bits for you.

Creating a master page

Working with a master page is very similar to designing and programming a regular Web page. You can drag and drop controls and work with events. Follow these steps to create a master page:

1. In Solution Explorer, right-click the project name and, from the context menu, choose Add New Item.
2. In the Templates area, select Master Page.
   The sorting of the templates in Add New Item puts Master Page before AJAX Master Page.
3. Open the master page in Source view, drag an Image control from the Toolbox and drop the control just after the default <div> tag and before <asp:ContentPlaceHolder>.
Although you can use Design view to drop controls into a master page, it’s easy to inadvertently put master content inside the `<asp:ContentPlaceHolder>` tags. If you find master content in the wrong place, you can move it where it belongs and grumble about why there’s no warning.

Figure 10-7 shows part of a master page in Design view. In the lower half of the figure are two `ContentPlaceHolder` controls (one of them is selected).

Adopting a master page while creating a regular page

When you add an ASP.NET page to your project, the IDE gives you the option of choosing its master page. To select a master page, follow these steps.

2. In the lower area of the Add New Item dialog box, select the Select Master Page check box.
3. Click Add.
4. Select the master page and then click OK.

The graphical environment prevents you from mucking up master page content while editing a regular page. Too bad the master page designer doesn’t protect the content placeholders the same way.
**Skinning Is Just What It Themes**

Themes and skins are another way of adding style and consistency to ASP.NET pages without having to deal with each control separately. You might use them as an alternative to converting a site to master pages. (See the previous section, “Using Master Pages with Slavish Devotion.”)

A *theme* is a unified look that you can apply to every page in your site. A *skin* is much the same, except that it applies to controls, such as the Button control and DropDownList control. This section introduces you to themes and skins with simple examples of a GoGreen theme that you can build on.

**Creating a theme for GoGreen**

ASP.NET looks for theme files in a special folder called App_Themes. Follow these steps to create the special folder, start a new theme, and design skins for two ASP.NET controls:

1. **In Solution Explorer, right-click the project name, and from the context menu, choose Add ASP.NET Folder ➪ Theme.**
   
   Visual Web Developer automatically creates the App_Themes folder and a subfolder called Theme1.

2. **Rename Theme1 to GoGreen, as shown in Figure 10-8.**

   ![Figure 10-8: Themes must reside in the special App_Themes folder.](image)

3. **Right click the GoGreen folder and add a skin file named Green.skin to the project (Add New Item ➪ Skin File ➪ Add).**
   
   The Green.skin file opens with the default skin template.

4. **At the bottom of the Green.skin file (after the --%> markup), enter the following skin description code:**

   ```xml
   --%>
   ```
5. Right click the GoGreen folder and add a style sheet named Green.css to the project (Add New Item ➪ Style Sheet ➪ Add).

6. In Green.css, add the following style sheet class:

```css
.greenstyle
{
    background-color:#ccffcc;
    font-style:italic;
    font-family:Segoe UI;
    font-size:xx-large;
}
```

You now have a skin for an ASP.NET DropDownList control and a Button control. The content of Green.skin declares that all controls of those types should use the CssClass called greenstyle. The greenstyle class is in Green.css.

Also in Green.skin, the Button control has been instructed to use a dotted border style (BorderStyle="Dotted"). This property isn’t part of the style sheet and therefore applies only to buttons and not to the drop-down lists.

There’s a catch to this: Nobody has told the controls in the site about the theme, or style sheet, or anything. That’s coming up in the next section.

### Assigning a theme to the whole Web site

The speedy way to tell every page and every control in a site to use a given theme is to post the notice in the web.config file. Follow these steps to make a theme available sitewide:

1. Open the web.config file.

2. Search within the `<system.web>` section for the element starting with `<pages`.

3. After the word `pages`, add the attribute and value combination:

```xml
<pages theme="GoGreen"/>
```
Every page in the site now knows to adopt the GoGreen theme at runtime. That signals to the DropDownList and Button controls within the pages to take on the GoGreen styles too. Unfortunately, the VWD designer doesn’t recognize the web.config setting and still shows non-green controls.

Depending on your needs, you may want to use the styleSheetTheme attribute instead of the theme attribute. The styleSheetTheme attribute allows local settings to override the global theme. Here’s the syntax:

```
<pages styleSheetTheme="GoGreen">
```

**Assigning a theme to an individual page**

You can set the theme for an individual page by configuring its Theme or StyleSheetTheme setting. This permits different themes on different pages. Here’s how to assign a theme to an individual page:

1. If you’ve assigned a theme in the web.config file, remove the assignment. (See the previous section.)
2. Open the ASP.NET page in Design view.
3. Click a blank area of the page, and open the Properties window for the Document object (F4).
4. Set the StyleSheetTheme property to the name of the theme (for example, GoGreen).

Drop a DropDownList and a Button control on the page and note how the IDE looks up the theme values and applies the colors in Design view.

If you leave the theme declaration in the web.config file, it takes effect sitewide. If you set it and forget it, you may end up confused as to why a given control or page won’t adopt a theme as instructed locally.
Chapter 11

Adding Navigation with TreeView, Menu, Breadcrumb, and SiteMap

In This Chapter

- Creating dynamic menus
- Building a Web.sitemap file for navigation
- Using the SiteMapDataSource with a TreeView
- Kneading breadcrumbs on your site

A navigation system provides a perceived structure to a site — even though every file could be crammed into the same physical directory. When a visitor feels lost, navigation controls provide a quick return to a known and comforting point.

This chapter walks you through the main ASP.NET navigation controls starting with TreeView, Menu, and SiteMapPath (breadcrumb). Don’t worry, you won’t get lost!

Using a Treeview on a Web Page

If you’ve used a computer, you know what a treeview is. It’s a compact way of presenting hierarchical information. A treeview is one of the most popular navigation components because it doesn’t overwhelm you. You can expand each leaf (also called a node) as required to discover more information and hide irrelevant parts.

Figure 11-1 shows that an ASP.NET TreeView looks more like an upside-down tree or a big root. In fact, it starts with a Root node and spreads out and down like a root. Whatever it is, this section gets to the root of using a treeview as a navigation device on a Web page.
Creating TreeView nodes in the designer

Visual Web Developer provides a graphical interface for building and nudging nodes on a treeview. Follow these instructions to configure a treeview on a Web page or master page:

1. From the Navigation section in the Toolbox, drag and drop a TreeView control onto an ASP.NET page.
2. Using the Smart Tag, open the Tasks menu and choose AutoFormat.
   The AutoFormat window appears.
3. Select a format (for example, News) and click OK.
4. From the Tasks menu, choose Edit Nodes.
   The TreeView Node Editor appears.
5. In the Nodes area, click the Add Root Node button.
   The Add Root Node button is on the far left in Figure 11-2.
6. In the Properties area, set the Text property to Home and the NavigateUrl property to default.aspx.
   The Value property matches the Text property unless you change it.
7. Add a child node by using the Add Child Node button (second from the left in Figure 11-2) and set the Text property to Login and the NavigateUrl property to login.aspx.
8. If you’re feeling ambitious, add more root and child nodes and experiment with reorganizing the nodes by using the four right-hand nudge buttons.
9. When you finish adding nodes, click OK.
At runtime, you can expand and collapse the child nodes, as shown in Figure 11-3. When you click the name of a page (for example, Login) the browser navigates to the (probably nonexistent) Login.aspx page.

You can add your own bitmaps to your project (like those in Figure 11-4) and tell the treeview to use them. Open the TreeView control’s Properties window (F4) and navigate down through LeafNodeStyle ➔ ImageUrl ➔ (ellipsis button) ➔ Select Image. Do the same again starting at RootNodeStyle and then HoverNodeStyle.
You may find it faster to create a few nodes by using the TreeView Node Editor and then switching to Source view. You can use copy and paste to assemble the remaining parts of the treeview. The following markup shows how the Login node nests inside the Home node.

```xml
<asp:TreeNode Text="Home" Value="Home"
    NavigateUrl="default.aspx">
<asp:TreeNode Text="Login" Value="Login"
    NavigateUrl="login.aspx">
</asp:TreeNode>
</asp:TreeNode>
```

Creating a **Web.sitemap** file for navigation data

An alternative to creating a treeview in an editor is to point the treeview to a data source, namely a **Web.sitemap** file.

Microsoft makes it easy to store and present your site’s structure by using a **Web.sitemap** file. The site map is a special XML file with a format that ASP.NET’s components recognize. Follow these steps to add and configure a **Web.sitemap** file:

1. **In Solution Explorer**, right-click the project name, and from the context menu, choose **Add New Item**.

2. **In the Add New Item dialog box**, select the Site Map icon (it may be far down in the list) and click **Add**.

   The **Web.sitemap** file opens in Source view.

3. **Edit the **Web.sitemap** file so it looks like the following markup**:

   ```xml
   <?xml version="1.0" encoding="utf-8" ?>
   <siteMap>
   <siteMapNode url="default.aspx"
        title="Home"
        description="Home Page">
   <siteMapNode url="login.aspx"
        title="Login"
        description="Log in or out of the site">
   </siteMapNode>
   </siteMap>
   
   For the most part, the data in the preceding XML matches the declarative **TreeNode** markup used in the preceding section, “Creating TreeView nodes in the designer.” The sitemap sample adds a **description** attribute (which provides a tooltip) and a corresponding value, such as **Home Page**.
XML code is finicky about upper- and lowercase characters. It’s safer to edit XML with a graphical editor. The Express version of Visual Web Developer doesn’t include a graphical XML editor, but there are many free editors. Search Microsoft’s Web site for *XML Notepad 2007* and you’ll find a link.

**Generating a treeview from a Web.sitemap file**

The `TreeView` control doesn’t consume the XML data in the `Web.sitemap` file directly. It needs an intermediary component, which by default, is the `SiteMapDataSource` control. The `SiteMapDataSource` control reads the `Web.sitemap` file, parses the data, and passes that data along to the `TreeView` in a format that the `TreeView` can use. Follow these steps to add the components and connect them to the `TreeView`:

1. If you haven’t already done so, add a `TreeView` control to your ASP.NET page. (See the earlier section, “Creating `TreeView` nodes in the designer,” if you need help.)

2. If you haven’t already done so, add a `Web.sitemap` file to your project.

   For sample data, see the preceding section, “Creating a `Web.sitemap` file for navigation data.”

3. With the ASP.NET page in Design view, from the Data category in the Toolbox, drag and drop a `SiteMapDataSource` control onto the page.

4. From the `TreeView` control’s Tasks menu, choose the `SiteMapDataSource` control, as shown in Figure 11-5.

At runtime, the treeview asks the `SiteMapDataSource` to relay the data found in the `Web.sitemap` file. You don’t need to specify the `Web.sitemap` file by name because the `SiteMapDataSource` uses that name by default.
Using the treeview with an XMLDataSource control

No rule says you must use the TreeView control only for navigation. It’s handy whenever you need to express hierarchical relationships. For example, Figure 11-6 shows three of the provinces of Canada, the name of the provincial capital, their flags, and their official language(s).

This section shows how to create the sample by using an XML file, the XMLDataSource control, and the TreeView.

Creating the XML file
The data source for this example is a small XML file. These steps show how to create the XML file and add the data:

1. In Solution Explorer, add an XML file called Canada.xml to your project (File ➪ New File ➪ XML File ➪ Add).

2. Add the following markup to the XML file:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<country name="Canada">
  <province en="Ontario" fr="Ontario">
    <capital en="Toronto" fr="Toronto"></capital>
    <flag url="on.gif"></flag>
    <ols en="none" fr="aucun" />
  </province>
  <province en="Quebec" fr="Québec">
    <capital en="Quebec City" fr="Québec"></capital>
    <flag url="qc.gif"></flag>
    <ols en="French" fr="Français"></ols>
  </province>
  <province en="New Brunswick" fr="Nouveau-Brunswick">
    <capital en="Fredericton" fr="Fredericton"></capital>
```
The preceding elements (for example, `<province>`) use the attributes `en` and `fr` to hold the English and French names. (English and French are Canada’s official languages; some provinces also have official languages.) The `<flag>` element uses a `url` attribute whose value is the name of a flag image.

**Configuring the XmlDataSource control**

The XmlDataSource control understands the structure of an XML file. In most cases, you only need to point it to the file’s location to make it work. Follow these steps to add and configure the XmlDataSource control.

1. From the Data section in the Toolbox, drag and drop an XmlDataSource control onto an ASP.NET page.
2. Open the Properties window (F4) for the XmlDataSource.
3. In the **DataFile** property, click the ellipsis button (...) to locate and select the `Canada.xml` file.
4. Click OK.

The preceding steps create declarative markup that looks like this:

```xml
<asp:XmlDataSource ID="XmlDataSource1" runat="server" DataFile="~/Canada.xml">
</asp:XmlDataSource>
```

In ASP.NET code and server-side markup, the tilde character (`~`) at the start of a file path tells ASP.NET to start looking for the file at the Web’s root folder and move down from there.

**Pointing the treeview to the XmlDataSource control and data**

Visual Web Developer includes a sophisticated designer that reads the nodes from the XmlDataSource and helps you select the data to display in the treeview. Follow these steps to configure the TreeView to use the XML data:

1. **Select the TreeView control and open its Tasks menu.**
2. **For the data source**, choose the XmlDataSource control (probably `XmlDataSource1`).
3. **Click Edit TreeNode Databindings.**

   The TreeView DataBindings Editor opens. The XML data nodes appear in the Available Data Bindings list (see Figure 11-7).
4. Select each node (except the Empty node) and click Add to add country, province, capital, flag, and ols to the Select Data Bindings list.

5. In the Select Data Bindings list, select country and for the TextField and ValueField properties, click name (not #Name).

6. One by one, from the Select Data Bindings list, select the province, capital, and ols nodes (skip flag) and set the TextField and ValueField properties to en.

7. In the Select Data Bindings list, click flag and set the ImageUrlField value to url.

If you don’t happen to have Canadian provincial flag images, you can download them from the book’s support site.

8. Clear the Auto-generate Data Bindings check box and click OK.

**Building a Menu for Your Site**

The ASP.NET Menu control works in two modes: static and dynamic. Static parts are always visible. Use them for main menu items along the top row, as shown in Figure 11-8. When items are dynamic, they appear only when triggered — usually by passing the mouse over the parent node. In Figure 11-8, Local, National, and World are dynamic items because they pop in and out (actually, they pop down and up).
Creating a menu in the designer

You can design an ASP.NET menu by using a graphical interface with prebuilt AutoFormat styles. You add the nodes and subnodes and nudge them into the appropriate location. Follow these steps to create a menu in the designer:

1. From the Navigation section in the Toolbox, drag and drop a Menu control onto the ASP.NET or master page.

2. To make a horizontal layout (refer to 11-8), open the Properties window and set the Orientation property to Horizontal.

3. Click the Smart Tasks button, and choose Edit Menu Items.

   The Menu Item Editor opens.

4. Using the toolbar buttons, add root and child items and configure the Text and NavigateUrl properties.

   The editing technique is almost identical to creating a TreeView control. If you need details, refer to the previous section, “Creating TreeView nodes in the designer.”

5. When you finish adding items, click OK.

6. Open the Properties window (F4), and set the fonts, colors, and style classes for the static and dynamic menu items.

   Listing 11-1 shows some example style information and markup for a horizontal menu.

Styling menus can be a lengthy and intricate task because you apply unique styles to the control as a whole, to submenus, menu items, selected items, and hover items. Throw in static and dynamic variants for all these and your head might be swirling.

Fortunately, item styles inherit characteristics from the parent item. For example, the hover styles automatically use the nonhover appearance unless you provide an override. In that case, you can bring the hover item to life by adding ForeColor="White", as shown in Listing 11-1.

Listing 11-1: Example Markup for a Horizontal Menu

```html
<style type="text/css">
.menuitem
{
    background: url(bgblue.png) repeat;
    height:22px
}
</style>
<!--...-->
```

(continued)
Listing 11-1: (continued)

```html
<asp:Menu ID="Menu1" runat="server"
Orientation="Horizontal" DynamicHorizontalOffset="2" Font-Names="Verdana"
Font-Size="Large" ForeColor="Black" StaticSubMenuIndent="0px"
StaticPopOutImageUrl="~/spacer.gif">
<StaticMenuStyle CssClass="menuitem" HorizontalPadding="10px" />
<StaticMenuItemStyle CssClass="menuitem" />
<DynamicHoverStyle ForeColor="White" />
<DynamicSelectedStyle CssClass="menuitem" />
<DynamicMenuItemStyle CssClass="menuitem" />
<StaticHoverStyle ForeColor="White" />
<items>
<asp:menuitem Text="News" Value="News">
<asp:menuitem Text="Local" Value="Local"></asp:menuitem>
<asp:menuitem Text="National" Value="National"></asp:menuitem>
<asp:menuitem Text="World" Value="World"></asp:menuitem>
</asp:menuitem>
<asp:menuitem Text="Sports" Value="Sports"></asp:menuitem>
<asp:menuitem Text="Weather" Value="Weather"></asp:menuitem>
</items>
</asp:Menu>
```

By default, the ASP.NET menu generates a black arrow to indicate that there are submenu items. You can supply your own image as the StaticPopOutImageUrl and DynamicPopOutImageUrl values. To hide the arrow, point the properties to a one-pixel, transparent GIF image.

### Generating a menu from a Web.sitemap file

The ASP.NET control can consume data from the Web.sitemap file. In fact, you can use the same Web.sitemap file for your site's TreeView, Menu, and breadcrumb controls. (Read about the breadcrumb — or SiteMapPath — control later in this chapter.)

Follow these steps to implement a Menu control based on a Web.sitemap file.

1. **Add a Web.sitemap file to your project (File ‒ New File ‒ Site Map).**
2. **Use the data in Listing 11-2 or create your own content for the Web.sitemap file.**
3. **Open an ASP.NET page in Design view, and from the Toolbox, in the Data category, add a SiteMapDataSource control to the page.**
4. **From the Toolbox, in the Navigation category, add a Menu control to the page.**
5. **From the Tasks menu, choose** SiteMapDataSource1.

6. **From the Tasks menu, choose Auto Format, apply a style, such as Simple, and then click OK.**

   If you browse to the page now, you see that News, Sports, and Weather are indented as child nodes of Home. You fix that in the next step.

7. **In the Menu control’s Properties window, set the** StaticSubMenuIndent **property value to 0px and the** StaticDisplayLevels **property value to 2.**

   By removing the indent, the static items appear to be at the same level in the site map hierarchy.

   Here’s a barebones version of the menu markup:

   ```
   <asp:Menu runat="server" ID="Menu2"
       StaticDisplayLevels="2"
       StaticSubMenuIndent="0px"
       DataSourceID="SiteMapDataSource1" />
   ```

   **Listing 11-2: Sample Data for a Web.sitemap File**

   ```xml
   <siteMap>
       <siteMapNode title="Home" url="default.aspx">
         <siteMapNode title="News" url="lcl.aspx" title="Local" />
         <siteMapNode url="ntnl.aspx" title="National" />
         <siteMapNode url="wrld.aspx" title="World" />
       </siteMapNode>
       <siteMapNode title="Sports" url="bbll.aspx" title="BaseBall" />
         <siteMapNode url="hcky.aspx" title="Hockey" />
         <siteMapNode url="sccr.aspx" title="Soccer" />
         <siteMapNode url="lsprts.aspx" title="Local" />
       </siteMapNode>
       <siteMapNode title="Weather" url="lfrcest.aspx" title="Local" />
         <siteMapNode url="wcaps.aspx" title="World" />
         <siteMapNode url="rcrds.aspx" title="Records" />
       </siteMapNode>
   </siteMap>
   ```

   The ASP.NET Menu control is a *templated* control. That means Microsoft leaves the formatting wide open for whatever markup you want to put into each menu item. The template markup doesn’t appear by default. To insert it, open the Tasks menu and choose Convert to DynamicItemTemplate and Convert to StaticItemTemplate. For details on working with templated controls, see Chapter 13.
Adding a Breadcrumb Feature to Your Pages

A breadcrumb (implemented through the ASP.NET SiteMapPath control) displays the hierarchy as a single line of hyperlinks. Web designers usually place it near the top of the page. While the user navigates, ASP.NET tracks the current location according to the filename and matches the filename with what it finds in the Web.sitemap file. It can then look backward in the hierarchy to provide a trail to the starting point, including the levels in between.

If the breadcrumb doesn’t show on a page, make sure that the page’s filename (for example, lcl.aspx) is somewhere in the Web.sitemap file. If the filename’s not there, it can’t be tracked, and the SiteMapPath control won’t have anything to display.

Creating a breadcrumb on a master page

This example puts the breadcrumb on a master page because it’s far more convenient than repeatedly adding navigation controls to individual pages in a site. Follow these steps to add a breadcrumb to a master page:

1. If you haven’t already done so, add a Web.sitemap file to your project (File ➤ New File ➤ Site Map) and add data from Listing 11-2.
2. Add a master page to your project (File ➤ New File ➤ Master Page).
3. From the Navigation section in the Toolbox, drag and drop a SiteMapPath control onto the master page.
4. Being sure to select the master page, add an ASP.NET page named lcl.aspx to your project.

The filename lcl.aspx is significant if you’re using the sample data from Listing 11-2 because lcl.aspx is part of that Web.sitemap file. If you’re using different data, choose a filename that is included in your Web.sitemap file.


As shown in Figure 11-9, the browser displays the Home node as a hyperlink and the remaining nodes (News and Local) as text.
Customizing a breadcrumb

You can customize several properties in addition to the fonts and colors. For example, you can replace the colon (:) between the nodes with another symbol by changing the value of the PathSeparator property. If you want the path to start at the current node on the left, change the PathDirection property to CurrentToRoot.

For advanced customization, open the Tasks menu to reach the control’s templates. You can go wild with different looks for the root node, path nodes, current node, and the path separator. To discover more about templates, turn to Chapter 13.
Chapter 12

Web Standards, Page Layout, and Usability

In This Chapter

- Creating standards-based pages
- Using CSS to create column layouts
- Trimming page bloat
- Making pages accessible and easy to use

It’s getting harder to create ugly, non-standard, and invalid HTML. The latest authoring tools work like a word processor to generate decent HTML markup on their own. Their syntax checkers flag problems with the parts that humans write. However, page creation is more than assembling a series of HTML tags. Putting a Web page together also involves some human engineering.

That’s where people — not machines — choose the best technologies for the intended use. By creating the most appealing layout and techniques, you make the pages easy for the target audience to use.

This chapter looks at adherence to technology standards, increasing usability, and some elements of page design that affect the quality of your ASP.NET pages.

Choosing an HTML Flavor

As Web pages and browsers become more sophisticated, content creators are paying more attention to standards. For example, it’s a sign of professionalism that a page declares the following in its markup:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```
That’s a declaration that the page complies with the XHTML 1.0 Strict standard. You still find many sites supporting the previous standard, HTML 4.01, or no standard at all. Too many pages claim in their document type declaration (DOCTYPE) to support XHTML or HTML 4 but don’t even come close when you dig into the markup.

XHTML 1.0 turns HTML markup into well-formed XML. The term well-formed means that the HTML code complies with XML rules such as these:

- **Self-closing tags:** Like a fire door, tags must be self-closing such that the `<img>` tag must look like `<img />` and `<br>` becomes `<br />`.
- **Close all tags:** Tags that formerly stood alone like `<p>` and `<li>` are completed with `</p>` and `</li>`.
- **Quote attributes:** Use `id="myquotedid"` instead of `id=myunquoteid`.
- **Consistent case:** Keep the starting and closing tags in the same case. You can’t use `<A>` for the opener and `</a>` for the closing tag. For the least hassle (and XHTML validation), just use lowercase everywhere. You can set that as an option in Visual Web Developer’s HTML editor.
- **Use entities:** Convert reserved characters to their special sequences (formally known as entities):
  ```
  < becomes &lt;
  > is &gt;
  & looks like &amp;
  ' comes off as &apos;
  " in code is &quot;
  ```

If you type `<>’"` in Design view and look at it in Source view, the editor *escapes* the text (that is, removes the reserved characters) to become `<&gt;&amp;#39;&quot;`. For the apostrophe, VWD prefers the character code version (`&#39;`) over the more readable `&apos;` entity.

**Visual Web Developer and standards**

By default, Visual Web Developer creates pages according to the XHTML 1.0 Transitional standard. Transitional isn’t nearly as rigid as the Strict flavor. However, Transitional is more acceptable for most sites because it permits legacy attributes in control tags. For example, where people now use `ID`, Transitional still supports `name`. These concessions ensure that JavaScript routines that rely on certain attributes continue to work. In other areas of the standard, relaxed rules ensure that pages look reasonable in older browsers that don’t know what to do with style sheets. You could argue that Transitional is a “light standard.” Sorry. Bad pun.
VWD helps you keep your markup faithful to your target standard. You can change the target in the HTML Source Editing toolbar (choose View ➪ Toolbars ➪ HTML Source Editing), as shown in Figure 12-1.

To see what validation does for you, try this:

1. Open a new ASP.NET page in Source view.
2. Set the validation target to XHTML 1.0 Transitional (refer to Figure 12-1).
3. Leaving the DOCTYPE declaration in place, change the HTML markup to the following, which mixes capital and lowercase characters:

   ```html
   <html>
   <head><title></title></head>
   <body></body>
   </html>
   ```

   Notice the squiggly lines under parts of the markup in Figure 12-2? These indicate problem areas.

4. Open the Error List window (View ➪ Error List).

   As shown in Figure 12-3, the list of errors includes unacceptable uppercase characters and a missing namespace.
5. Delete the entire DOCTYPE declaration from the page.

5. Delete the entire DOCTYPE declaration from the page.

The list of errors doesn’t change. The VWD validation is based on the schema setting in the HTML Source Editing toolbar and not on the optional declaration of a document type.

If you’re targeting a standard other than XHTML 1.0 Transitional, you might find it annoying to keep selecting your target. You can change the default validation target by choosing Tools ➪ Options ➪ Text Editor ➪ HTML ➪ Validation. (If you don’t see these options, check the Show All Settings box in the Options dialog box.)

**External XHTML validation**

After you deploy your site on the Internet, you can confirm that your pages are XHTML compliant. The World Wide Web Consortium (W3C) provides a validation service at http://validator.w3.org. You provide the URL of your page to get a report that looks like Figure 12-4.

You can advertise your site’s XHTML status by including the W3C’s logo and a link to the validator. The following markup inserts the validation icon and link on a page:

```
<a href="http://validator.w3.org/check?uri=referer">
<img alt="Valid XHTML 1.0 Transitional" height="31"
src="http://www.w3.org/Icons/valid-xhtml10" width="88" />
</a>
```

### Creating Columns Using CSS Float

Professional Web page designers usually divide a page into three or four major sections: a masthead along the top, a content area in the middle, and
a footer along the bottom. Then they subdivide each area. The masthead might include a banner, an advertisement, a menu, and breadcrumbs. Likewise, designers often split the content area into two or three columns.

The old way to hold this structure in place was to erect a huge, complicated HTML table with lots of merged cells, spanned columns, and tricky rows. The technique still works but is on the way out (deprecated in geekspeak) for several reasons:

- HTML table tags are intended to present tabular data in a meaningful way, not to lay out page structure.
- People with disabilities find it harder to navigate and interpret content when you spread it across table cells.
- Browsers are slower to render large, complicated tables.

Microsoft provides a good example of CSS layout in its free Small Business Starter Kit, shown in Figure 12-5 as the fictional Fabrikam corporation site. You can download the sample from www.asp.net. The following sections explore some of the significant layout features.

![Figure 12-4: Checking for valid XHTML at validator.w3.org.](image)
Divvy up the page with `<div>` tags

The following code shows the barebones structure of the main page. The page uses the `<div>` tag extensively to create the top information, the content container, and footer. (Actually, it has a footer with a subfooter). The nested structure creates columns using `<div>` tags.

```html
<form id="form1" runat="server">
  <div id="top-information">
    <div id="logo"></div>
    <div id="top-information-home"></div>
    <div id="top-information-phone"></div>
  </div>
  <div id="none"></div>
  <div id="nav-main"></div>
  <div id="poster-photo-container">
    <div id="feature-area-home"></div>
  </div>
  <div id="content-container-two-column">
    <div id="content-main-two-column"></div>
  </div>
</form>
```

Figure 12-5: Microsoft’s Small Business Starter Kit uses CSS column layout rather than an old-fashioned Table layout.
Document Outline lays out the structure

Visual Web Developer’s Document Outline pane (View ➪ Document Outline) makes it easy to analyze the layout structure. Figure 12-6 shows that the page contains three subsections that include the logo, the home link, and the phone number.

Dedicated style rules and float: left

Each <div> tag has an ID attribute that corresponds to a dedicated style rule selector. For example, the tag

```html
<div id="content-main-two-column">
</div>
```

receives its style information — notably width and float — from the following style rule:

```css
#content-main-two-column
{ float: left; width: 540px; }
```

The use of float: left is critical to creating columns using CSS layouts. In this case, the <div> box is 540 pixels wide and floats to the left. The float forces other content to flow to its right, starting at the top.
You can use the ASP.NET Panel control to generate `<div>` tags for layout. Use class-based style rules — the ones that start with a dot (.) — and set the Panel control’sCssClass property to the style’s name.

Reducing Load Times and Improving Performance

As Web pages become busier, performance starts to drag. If a page seems slow on your workstation, expect it to get worse when posted to the Web server. While you can’t always control the speed of the Internet connection or the Web server’s horsepower, you can certainly do your part to cut page load times and improve the overall throughput. This section looks at two techniques: reducing ViewState and employing caching.

Turning off ViewState

ASP.NET server controls make programming easier, but they make Web pages fatter. If you’re not careful, ASP.NET can bloat a relatively small page by overdoing a hidden feature called ViewState. Look at the output of any ASP.NET page and you find ViewState. It resembles the following (much abbreviated) markup when you look at the HTML in the browser (View → Source):

```html
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUL...80sXAJunhsp3aKX6m/d" />
```

ViewState is how ASP.NET tracks the current settings for all its controls. You see an ordinary, hidden `<input>` tag with a string of nonsense characters as the value attribute. That value is the Web server’s personal aide-memoire as to what the page was like when it sent the HTML to the browser.

That blob of overhead travels back and forth (it roundtrips in geekspeak) for no good reason. If there’s nothing in the server-side code that can change the text in a Label control or GridView, why track the ViewState?

You can turn off ViewState for an individual control (for instance, a TextBox) by setting its EnableViewState property to False.

If you’re confident that none of the controls on a page will change, you can turn off ViewState for the entire page. In Source view, at the top of the page, add the part that you see shown in bold:

```html
<%@ Page Language="VB" EnableViewState="false" %>
```
Going to extremes, if you’re positive that you don’t need ViewState anywhere in the whole ASP.NET site, you can switch it off in the web.config file. Look for the <page> element (somewhere within the <system.web> element) and add the following part shown in bold:

```xml
<pages enableViewState="false"/>
```

Even when you turn off ViewState for an entire site, ASP.NET doesn’t give up completely. Some remnants remain in the page — as if it’s using ViewState to track the fact that ViewState is off.

**Caching “expensive” content**

Imagine poor old Internet Information Services grinding away to produce an ASP.NET page full of elaborate three-dimensional charts. The data for the charts comes from a SQL Server database that executes some mind-numbingly complicated queries.

Eventually, the HTML, images, and associated JavaScript are assembled and shipped across the Internet to the browser. Congratulations on a job well done. In the next fraction of a second, someone else requests exactly the same page. The Web server starts all over again, building the shading and gradients on the charts and re-requesting the data. It’s a waste of time and, in geekspeak, expensive.

To reduce the load on the server, you can store a copy of a complicated diagram, image, report, or page in the server’s memory. On the next identical request, the Web server looks for that content in the cache. If it finds something usable, it sends the cached copy.

**Page-level caching**

When you identify a page that would benefit from caching, open the page in Source view and, at the top of the page, add the following directive:

```csharp
<%@ OutputCache Duration="300" VaryByParam="none" %>
```

The preceding line of code gives these instructions to the Web server:

Create this page as usual on the occasion that someone requests it. Hold the page in memory for 300 seconds (five minutes) and send your cached copy if anyone else asks for it. Yes, send the cached version even if the data in the database has changed. After five minutes, dump the cached version and start over with a new page if asked.
Caching is powerful, but it can drive you crazy if you’re not aware of it. Try this experiment to see just how strange things can get:

1. Drop a Label control and Button control on an ASP.NET page.
2. In Source view, configure the markup as follows:

   ```
   <asp:Label ID="Label1" runat="server">
   <% =Now.ToLongTimeString%></asp:Label><br />
   <asp:Button ID="Button1" runat="server" Text="Button" />
   ```

   The `<%=Now.ToLongTimeString%>` inserts the time as text for the Label control.
3. Run the page and click the button several times.
   The time updates on every button click, as you’d expect.
4. Add the following code to the very top of the ASP.NET page.

   ```
   <%@ OutputCache Duration="300" VaryByParam="none" %>
   ```
5. Run the page and click the button.
   The page updates as before.
6. Click again, several times.
   Time stands still for five minutes — at least according to the server.

On the first click, the Web server updates the page and then caches the content. On subsequent clicks, it sends the cached version.

If you’re having problems with a page that doesn’t update properly, check whether it’s using caching. It may be that you’re viewing a cached version.

**Cache is not forever**

Although you can ask a Web server to cache pages, there’s no guarantee that the server keeps the cache for the set time — or caches it at all. When the Web server runs short of memory (because the cache is full or there are severe demands on its memory), it dumps items from the cache automatically. If your pages are running on a shared server, it’s possible that someone else’s pages gobble up the available cache and your performance deteriorates unexpectedly.

**Meeting Accessibility Requirements**

Increasingly, Web sites created for government agencies must meet accessibility requirements. These standards ensure that the site is usable by people who have difficulty seeing, hearing, moving, or processing images. The goal is
to generate online documents that specialized browsers, such as screen readers and braille displays, can handle easily.

In the United States, people often refer to Section 508 of the Rehabilitation Act of 1973, more commonly known as Section 508. In 1999, the W3C produced its Web Content Accessibility Guidelines 1.0 (WCAG 1.0). You can read the document at www.w3.org/TR/WAI-WEBCONTENT/.

Accessibility and usability have many issues in common. See the upcoming section, “Increasing a Page’s Usability,” for ways to make pages easier to use for all visitors.

**Alternate text for images**

For the most part, ASP.NET server controls meet accessibility requirements, but a great deal depends on the page developer. For example, you wouldn’t want to create problems for blind — or colorblind — readers by instructing them to click a red or a blue image to continue.

A key requirement for accessibility is a text equivalent for images and multimedia content. The ASP.NET Image, ImageButton, and ImageMap controls include an AlternateText property in which you can enter an explanation of the image. If your layout uses images for padding and fillers, you can set the GenerateEmptyAlternateText property to true so the controls generate empty strings (that is, quotes with nothing inside them). This way, accessibility checker software won’t report those images as missing alternate text, and specialized readers won’t consider them as unimportant.

**Avoiding output as tables**

Some ASP.NET controls, such as the Menu control, use the HTML `<table>` tag to create their structure. Accessibility guidelines frown on tables except for tabular data.

Fortunately, many ASP.NET controls are templated, which means that you’re free to design their markup and replace undesirable `<table>` tags with style sheet–friendly `<div>` tags. For more information on using templated controls, see Chapter 12.
**Is client script allowed?**

Several ASP.NET controls use client-side JavaScript to trigger postbacks or validate data. For example, the DropDownList control’s AutoPostBack property is convenient for the majority, but not so accessible. Be sure to check the accessibility requirements of your project because some standards do allow for client-side script.

Additionally, you can disable much of the client-side JavaScript using properties like `EnableClientScript="false"` in the validation controls.

**Validating Web accessibility**

The full version of Visual Web Developer (not Express) includes a Check Page for Accessibility feature. By default, the button appears on the HTML Source Editing toolbar. When you start the checker, it asks for the desired validation standards, as shown in Figure 12-7. The results of the validation appear as warnings in the Error List pane.

There are hundreds of free tools to check the accessibility of Web pages. They cover the U.S. legislative requirements, foreign standards, and the W3C’s WCAG 1.0 recommendation. You can view an extensive list at www.w3.org/WAI/ER/tools/complete/.

**Increasing a Page’s Usability**

Power users make extensive use of the keyboard. Some touch the mouse only when necessary — because the mouse slows them down or overuse of the pointer has left them with repetitive strain injury (RSI). You can make any data entry form more user-friendly by paying attention to tab order, accelerator keys, and first focus.
Setting the tab order

Data entry professionals move from control to control using the Tab key. They also care about the order of the controls on a form. Consider this: You’re a customer service representative taking information from someone on the telephone. You ask for the order details in a logical sequence and enter the data. However, each time you hit the Tab key, the focus goes to the wrong control.

ASP.NET controls include a TabIndex property that lets you set the tab order. A tab sequence goes from the lowest to highest index number and then starts over. It’s best to check with the users as to their preference in tab order. For example, in Figure 12-8, should the tab order be horizontal from Produce to Quantity or vertical from Produce to UPC? The choice could depend on the order in which the user is gathering the data or the behavior of a familiar Windows application.

You never have full control over the browse sequence because browsers themselves include controls, such as the Address text box in the tab sequence.

Enter the TabIndex value as 10, 15, 20, 25 and so on rather than 1, 2, 3, 4. If you have to insert a control later or move controls around, you won’t need to renumber the TabIndex property on as many controls.

Adding access/accelerator/shortcut keys

Access keys (also called accelerator keys and shortcut keys) let users jump to a control directly by way of a keyboard sequence rather than moving the mouse. If you squint hard enough at the word Disg in Figure 12-8, you note that the lowercase “c” is underlined. The convention in Windows is to
underline the letter that serves as the shortcut key, in this case Alt+C. You can implement the same effect in ASP.NET by setting the `AccessKey` property as shown in the following steps:

1. Add a `Label` control with the ID `lblDisc` to an ASP.NET page.
2. Add a `TextBox` control with the ID `txtDiscount` to the page.
3. In the `Label` control’s Properties window (F4), set the `AccessKey` property to C, the `AssociatedControlID` property to `txtDiscount`, and the `Text` property to `Dis<u>c</u>%:`.

At runtime, the key sequence Alt+C moves the focus to the text box.

Test your access keys in the target browser to be sure that they aren’t reserved for use by the browser. You don’t want the browser to close when the user is intending to submit a screen full of data!

### Setting the focus on startup and default buttons

There’s usually a logical starting point for entering data in a Web page. Faced with a form to fill in, most of us probably look to the upper-left corner. Unfortunately, browsers force users to tab or move the mouse to enter data.

To make data entry easier, ASP.NET forms let you set a default control. In the `Form` control’s property page, set the `DefaultFocus` property to the name of the starting control, such as `txtFirstName`. After the page loads, the cursor jumps to the `txtFirstName` text box automatically.

The `Form` control also supports a `DefaultButton` property to submit the form when the user presses the Enter key. At runtime, the page inserts a JavaScript sequence inside the `<form>`:

```html
onkeypress="javascript:return WebForm_FireDefaultButton(event, 'Button1')"
```

Some page designers make the Cancel button the default in cases where an unintentional keystroke has serious consequences — like wiping out tons of unrecoverable data.
Chapter 13

Designing the ListView and Other Templated Controls

In This Chapter

- Understanding templates in databound controls
- Building ListView templates by hand
- Paging with the DataPager control

ASP.NET ships with a good selection of databound controls, including the GridView, DataGrid (yes, it’s still there), DetailsView, FormView, DataList, Repeater, and ListView. The advanced controls include AutoFormat wizards that apply basic schemes and layouts, often by customizing built-in templates. These templates determine how the control renders specific elements, such as the header, data rows, alternate rows, and the footer. This chapter tells you what you need to know about templates so that you can produce the look and behavior you want.

This project uses Julie’s DVD database from Chapter 3. The database has very few fields, making the steps shorter. If you don’t want to create the database manually, you can download it and the SQL script from this book’s Web site. Actually, you can use any SQL Server database you want, but you need to adjust details, such as field names.

Understanding Templated Controls

Templated controls let you insert custom text, markup, styles, and other ASP.NET controls into a control’s basic framework. While altering the appearance and function, you still take advantage of the control’s built-in ability to bind to data and loop through items.

Templates are modular elements in that the databound control swaps template sections in and out according to its needs. For example, when the control needs to accept user input, it displays a specialized editing template.
For brevity, the instructions in this chapter don’t validate user input or cover performance-enhancing AJAX. (See Chapter 19 for more on validation; Chapters 4 and 15 talk more about AJAX.)

Repeating yourself with the Repeater

The ASP.NET Repeater control is the least-complicated templated control. In Listing 13-1, you see its template names, such as <HeaderTemplate>, <ItemTemplate>, and <SeparatorTemplate>. Your job is to put HTML markup inside the template areas. For example, the markup within the <SeparatorTemplate> is the horizontal rule (<hr />).

If a template name has the word Item in it, it usually repeats for each item of data. For the rule to hold, <SeparatorTemplate> should be <Separator ItemTemplate>.

Listing 13-1: Template Markup for a Barebones Repeater Control

```xml
<asp:Repeater ID="Repeater1" runat="server" DataSourceID="LinqDataSource1">
  <HeaderTemplate>
    <b>The HeaderTemplate</b><br />
  </HeaderTemplate>
  <ItemTemplate>
    The ItemTemplate<br />
  </ItemTemplate>
  <AlternatingItemTemplate>
    <i>The AlternatingItemTemplate</i><br />
  </AlternatingItemTemplate>
  <SeparatorTemplate>
    <hr />
  </SeparatorTemplate>
  <FooterTemplate>
    FooterTemplate</FooterTemplate>
</asp:Repeater>
```

Here’s what you need to know about Listing 13-1:

→1-5 When a Repeater binds to data, it displays the content within the <HeaderTemplate> once, no matter how many items in the data.

→6-12 Then for each data item, Repeater displays the content of the <ItemTemplate>, <AlternatingItemTemplate> and <SeparatorTemplate> templates.

→15-17 The <FooterTemplate> content appears only once, after the last item of data.
Figure 13-1 shows Listing 13-1 as rendered in the browser. There are five data items.

Although Figure 13-1 is bound to data, it doesn’t display any data. The templates include only static content. To display data, you use inline ASP.NET code like this:

```csharp
<%#Eval("CategoryName")%>
```

This odd-looking syntax tells ASP.NET, “Go to the data source, look for the CategoryName column for the current item, and insert the value you find right here.” When you combine the inline statement with the template command and markup, it looks like this:

```xml
<ItemTemplate>
    <%#Eval("CategoryName")%><br />
</ItemTemplate>
```

You can put ASP.NET controls inside templates. For example, you can embed a Label control as an alternate way to display the same text:

```xml
<AlternatingItemTemplate>
    <i>
        <asp:Label ID="lblAltItem" runat="server"
            Text='<%#Eval("CategoryName")%>' />
    </i><br />
</AlternatingItemTemplate>
```

When an embedded control requires quotes around an inline statement (see the preceding example), use single quotes (‘) on the outside and maintain the double quotes (“) within the statement.
Letting the designers generate templates

As much as possible, this book uses Visual Web Developer’s graphical designers to work with templates. The designers understand and write template syntax fluently, so you don’t need to master it. As shown in Figure 13-2, the GridView control’s designer supports templates for each column after you convert the column into an `<asp:TemplateField>`.

The Repeater control example displays all its templates as long as it has data to render. The GridView and other advanced controls support an additional template, `<EditItemTemplate>`, for editing. `<EditItemTemplate>` uses input controls, such as the TextBox, DropDownList, or RadioButton, to insert and update data.

When you click the Edit button in a templated GridView control, the selected line changes dramatically by hiding the `<ItemTemplate>` for the row and displaying the `<EditItemTemplate>` in its place. Here’s an example of a TextBox control for editing data within a template:

```xml
<EditItemTemplate>
  <asp:TextBox ID="TextBox1" runat="server"
              Text='<%# Bind("ProductName") %>'>
  </asp:TextBox>
</EditItemTemplate>
```

Notice that the preceding example uses Bind() rather than Eval() in the inline code. Controls that read data and write back to the data source, which is two-way binding, use the Bind() method. The GridView designer inserts Bind() automatically when generating a field that can be updated.
The highly capable GridView doesn’t support the addition of new rows. Insertion is often left to the FormView control, as shown in Figure 13-3. The FormView control’s <InsertItemTemplate> looks like the following:

```xml
<InsertItemTemplate>
  ProductName:
  <asp:TextBox ID="ProductNameTextBox" runat="server"
    Text='<%# Bind("ProductName") %>' />
  ...
</InsertItemTemplate>
```

Figure 13-3: The FormView supports inserting with <InsertItemTemplate>.
Rolling Your Own with the ListView Control

Some developers felt that the ASP.NET data controls were, er, too controlling. For example, programmers like the way the GridView handles data editing, deleting, and paging, but (and there’s always a “but!”) they complain that it locks the data inside HTML tables.

The ASP.NET team responded with the ListView control, which doesn’t dictate the HTML. It’s up to you to create the HTML markup — prepare to get your hands dirty!

This section walks you through the creation of the data-driven page shown in Figure 13-4. You create CSS-friendly markup (no tables) for the ListView.

Generating the DataContext

The DataContext is a set of data classes that let ASP.NET code believe that it’s dealing with objects rather than database tables and fields. A VWD designer maps objects to database tables. You make quick work of generating the DataContext by following these steps:

1. In Solution Explorer (View ➤ Solution Explorer) add the JulieDVD.mdf database (created and configured in Chapter 3) to the project’s App_Data folder.
2. Add a LINQ to SQL Classes file named juliедvdataclasses.dbml (File ➤ New File ➤ LINQ to SQL Classes ➤ Add) to the project.

Visual Web Designer offers to create the App_Code folder for you. By all means, let it do the work!

3. In Database/Server Explorer (View ➤ Database/Server Explorer), expand the Data Connections and JulieDVD.mdf database nodes until you locate the Movies node (it’s under Tables).

4. Drag the Movies node and drop it on the left-hand pane of the Object Relational Designer.

The O/R Designer names objects by using the singular of the table name. Unfortunately, the designer doesn’t realize how ridiculous the English language can be. It thinks that because the singular of Categories is Category, the singular of Movies is Movy! Don’t rename it — it’s a conversation piece!

If you’re curious about what’s in the DataContext, expand all the nodes in App_Code and open juliедvdataclasses.designer.vb.

**Configuring the LinqDataSource**

The LinqDataSource control is the intermediary between the ListView control and the DataContext. In the spirit of digging into the markup in this chapter, the following steps show how to write the LinqDataSource control attributes by hand.

1. Add an ASP.NET page named julielistview.aspx to your project (File ➤ New File ➤ Web Form ➤ Add) and open it in Source view.

2. From the Toolbox, drag a LinqDataSource control and drop it just before the closing </form> tag.

3. Configure the LinqDataSource control to allow deleting, inserting and updating by referencing the DataContext and Movies table that you created in the preceding section.

The configured LinqDataSource should look like this:

```xml
<asp:LinqDataSource ID="LinqDataSource1" runat="server"
ContextTypeName="juliedvdataclassesDataContext"
EnableDelete="True" EnableInsert="True"
EnableUpdate="True" TableName="Movies">
</asp:LinqDataSource>
```

If you explore juliедvdataclasses.designer.vb, you find code that represents the Movies collection starting with this:

```xml
<Table(Name:="dbo.Movies")>
```
Setting up the ListView

You’re almost ready to design the ListView control. The only thing missing is the, er, ListView control. Follow these steps to add and configure the control on an ASP.NET page:

1. Open julielistview.aspx in Source view and, from the Toolbox, drag and drop a ListView control inside the default <div></div> tags.

2. Configure the ListView so that the DataKeyNames property uses ID (the movie ID), it gets its data from the LinqDataSource that you added (see preceding section) and the template to insert new items appears at the end of the list.

Here’s how the code looks when complete:

```xml
<asp:ListView ID="ListView1" runat="server"
    DataKeyNames="ID" DataSourceID="LinqDataSource1"
    InsertItemPosition="LastItem">
</asp:ListView>
```

Adding the mandatory LayoutTemplate

The ListView control must include a <LayoutTemplate> section. The LayoutTemplate is where you define the outer structure or container of the ListView control. The structure reminds me of a Tim Horton’s donut (not the kind with the hole, the solid ones). Inside the donut is a reserved cavity, or placeholder, where talented sous-chefs insert the delicious cream or jelly. (Boston cream for me, please.)

At runtime, ASP.NET squirts (that’s the technical term) content from other templates (such as the ItemTemplate) into the designated placeholder. Here are the guiding rules for the LayoutTemplate:

- **The LayoutTemplate must include a placeholder control:** This control can be a <div>, a <tr>, or any other element that acts as a container.
- **The placeholder control must use itemPlaceholder as its ID:** By default, the ListView looks for itemPlaceholder as the place to put content generated by other templates.
- **The placeholder control must have runat="server":** This code makes an ordinary HTML element a server control that can accept server-side content.
After you digest the rules, follow these steps to add and configure the layout template:

1. **Open julielistview.aspx in Source view.**
2. **Add the following LayoutTemplate markup inside the ListView control (just before the closing </asp:ListView> is good):**

   ```xml
   <LayoutTemplate>
   <ul class="MovieList">
     <asp:PlaceHolder ID="itemPlaceholder" runat="server" />
   </ul>
   </LayoutTemplate>
   ```

   The preceding markup establishes that all injected content resides between `<ul>` tags and inside the `<asp:PlaceHolder>`. Yup, you’re building an unordered list with `<li>` tags. Don’t worry! It won’t look like a bulleted list by the time you’re finished.

### Displaying data with ItemTemplate

The `<ItemTemplate>` template generates one item for each record in the data source. The ItemTemplate content is usually read only. The layout doesn’t need to be horizontal because you’re using a gridlike structure. Figure 13-5 shows a single row of data (one movie) with the title, description, date, and images displayed vertically in a box. (You can download the images from the book’s Web site.)

![Figure 13-5: A single item generated by Item Template.](image)

Follow these steps to create the markup and data for an ItemTemplate:

1. **Open julielistview.aspx in Source view.**
2. **After the closing </LayoutTemplate> element (but before </asp:ListView>), insert the following markup:**

   ```xml
   ...
Notice that the content of the ItemTemplate is contained in a list item (li) tag. You use sets of <div> tags to hold the title, description, and date. At runtime, the Eval() method substitutes values from the current database row.

Most of the preceding markup generates image buttons for editing and deleting items. Clicking the Edit button tells the ListView control to switch to the <EditItemTemplate> template, which is what you build in the next section.

ASP.NET doesn’t care about the order of the templates within the ListView control. This walk-through examines the templates in a logical sequence, but you may find that the built-in template designers mix up the template order.

The optional <AlternatingItemTemplate> works exactly like the <ItemTemplate> except that it acts on every other item. You can use it to provide an alternating or contrasting visual appearance for items. The ListView uses the <ItemTemplate> if there’s no <AlternatingItemTemplate>.

Editing records with EditItemTemplate

The <EditItemTemplate> template defines the appearance of an item while in editing mode. That means it displays input controls, such as text boxes, drop-down lists, and check boxes. Buttons update or cancel the operation. Figure 13-6 shows an item in edit mode.
Follow these steps to create the markup and data for an `EditItemTemplate`:

1. **Open** `julielistview.aspx` **in Source view.**
2. **After the closing** `<ItemTemplate>` **element (but before** `<asp:ListView>`) **insert the following markup:**

   ```xml
   <EditItemTemplate>
   <li>
   <div class="OneMovie">
       <div class="MovieName">
           <asp:TextBox ID="TitleTextBox" Text='<%# Bind("Title") %>'
              ToolTip="Movie Title" runat="server" />
       </div>
       <div class="MovieDetail">
           <asp:TextBox ToolTip="Description" TextMode="MultiLine"
              Rows="2" ID="DescriptionTextBox" runat="server"
              Text='<%# Bind("Description") %>' />
           <asp:TextBox ID="DateAddedTextBox" runat="server" ToolTip="Date Added" Text='<%# Bind("DateAdded") %>' />
           <asp:ImageButton ID="UpdateButton" AlternateText="Update"
               CommandName="Update" runat="server"
               ImageUrl="/Images/update.gif" />
           <asp:ImageButton ID="CancelButton" AlternateText="Cancel"
               CommandName="Cancel" runat="server"
               ImageUrl="/Images/cancel.gif" />
       </div>
   </div>
   </li>
   </EditItemTemplate>
   ```

   Notice in the preceding markup that the inline code for the `TextBox` controls includes the `Bind()` method rather than `Eval()`. `Bind()` supports two-way binding, which means it displays values and writes them back to the data source.

   The **Update** and **Cancel** buttons (**ImageButton controls here**) include a `CommandName` property set to `Update` and `Cancel` respectively. `CommandName` tells the `ListView` which button was clicked.
Adding records with InsertItemTemplate

The ListView control supports inserting new records at the beginning or end of the list — provided that you include an <InsertItemTemplate> section with the appropriate controls. InsertItemTemplate (shown in Figure 13-7) is almost identical to EditItemTemplate (see preceding section) in that it uses the same input controls. The main difference is in the use of an Insert button.

Follow these steps to create the markup and data for an InsertItemTemplate:

1. Open julielistview.aspx in Source view.
2. After the closing </EditItemTemplate> element (but before </asp:ListView>), insert the following markup:

```xml
<InsertItemTemplate>
  <li>
    <div class="OneMovie">
      <div class="MovieDetail">
        <asp:TextBox ID="TitleTextBox" runat="server"
                      Text='<%# Bind("Title") %>' />
      </div>
      <div class="MovieDetail">
        <asp:TextBox ID="DescriptionTextBox" runat="server"
                      Text='<%# Bind("Description") %>' />
        <br />
        <asp:TextBox ID="DateAddedTextBox" runat="server"
                      Text='<%# Bind("DateAdded") %>' />
        <br />
        <asp:ImageButton ID="InsertButton" AlternateText="Insert"
                         CommandName="Insert" ImageUrl="~/Images/update.gif" runat="server" />
        <asp:ImageButton ID="CancelButton" AlternateText="Cancel Insert"
                         CommandName="Cancel" ImageUrl="~/Images/cancel.gif" runat="server" />
      </div>
    </div>
  </li>
</InsertItemTemplate>
```

Figure 13-7: Adding a movie with the InsertItemTemplate.
The key for inserting a record is using `CommandName="Insert"` in the `Button` control so that the `ListView` control knows what the user wants to happen.

If your markup in Source view looks messy, choose `Edit` ➪ `Format Document` or the press Ctrl+K and then Ctrl+D to reformat it.

**Advising users there’s no data with EmptyDataTemplate**

The `<EmptyDataTemplate>` contains the markup that tells users that the `ListView` has no data to display. Unlike the `<ItemTemplate>`, `<EditItemTemplate>`, and `<InsertItemTemplate>`, the `<EmptyDataTemplate>` control doesn’t push its content through the placeholder inside the `<LayoutTemplate>` section. It uses its own container.

Users may never see the `<EmptyDataTemplate>` — even if the `ListView` has no data! If you configure the `ListView` to display the `<InsertItemTemplate>` content (by setting `InsertItemPosition` value to `LastItem` or `FirstItem`, the “No data” message never appears).

Figure 13-8 shows a runtime example of an `EmptyDataTemplate` in action. To create the template, follow these steps:

1. **Open** `julielistview.aspx` in Source view.

2. **After the closing`</InsertItemTemplate>` element (but before`</asp:ListView>`), insert the following markup:

   ```xml
   <EmptyDataTemplate>
   <ul class="MovieList">
   <li>
   <div class="OneMovie">
   <div class="MovieName">
   No data was returned.<br />
   </div>
   </div>
   </li>
   </ul>
   </EmptyDataTemplate>
   ```

   The `<EmptyDataTemplate>` doesn’t use the `<LayoutTemplate>` as its container and therefore needs to provide its own container, such as `<ul></ul>`, in this sample.
To test the **no data** message at runtime, follow these steps:

1. **Configure the data source so that no items are returned.**
   
   You can do so by wiping all the records out of the database or by creating a select statement that you know won’t return results.

2. **Set the ListView control’s InsertItemPosition property value to None.**
   
   This step is necessary because the ListView ignores the EmptyDataTemplate when InsertItemTemplate is enabled.

### Using the ItemSeparatorTemplate

Although not required for this example, the `<ItemSeparatorTemplate>` generates the markup that goes between items. The separator usually delimits rows in a grid. The following markup adds a vertical block between items in the sample project:

```xml
<ItemSeparatorTemplate>
  <li>
    <div class="OneMovie"
      style="width:1px; background-color:Silver"> 
    </div>
  </li>
</ItemSeparatorTemplate>
```

### Making a horizontal list with flow

If you run the code in this section without the styles, you see one long, odd-looking bulleted list. The page is functional in that you can edit and delete items. But, without a better layout, who would want to use it?
Two style rules, MovieList and OneMovie, are critical to the layout because they handle the alignment and remove the default list item bullets. Follow these steps to add the basic formatting to the ListView elements:

1. **Open julielistview.aspx in Source view.**
2. **In the <style></style> area (add the tags if they don’t exist), add the following style class for the MovieList and <li> tag:**

   ```
   .MovieList li
   {
     display: inline; float: left;
     margin-bottom: 15px; margin-left: 15px;
   }
   ```

   The preceding uses display:inline and float:left to change the list items (<li>) from a vertical to a horizontal display.

3. **Add a class named OneMovie to create a 100-pixel by 175-pixel box-like container out of the <div> tags embedded in the <li> item:**

   ```
   .OneMovie
   {
     background: #b9d3ee; border: #648abd 2px solid;
     color: maroon; font-size: small;
     height: 100px; padding: 4px;
     width: 175px;
   }
   ```

   With the preceding code, the list items adjust and flow as you resize the browser. If you’d rather constrain the items to a fixed width, put the ListView control inside a container tag:

   ```
   <div style="width: 680px; border: solid 1px black;
     padding-bottom: 10px">
   </div>
   ```

   I haven’t shown the remaining style classes because they don’t control the layout. You can download them from the book’s Web site.

### Using the DataPager with a ListView

The ASP.NET DataPager control lets users deal with data in small chunks by stepping through it one page at a time. You can set the number of items per page and the pager appearance. The built-in navigation options are Next/Previous buttons and numeric paging. Follow these steps to add paging to this chapter’s ListView sample:
1. Open julielistview.aspx in Source view.

2. From the Toolbox, drag a DataPager control from the Data section and drop it before the closing </form> tag.

3. In the DataPager control’s Properties window (F4), in the Fields property, click the ellipsis button (...).
   The Fields dialog box appears.

4. In the Available Fields area, click Numeric Pager Field, click Add, and then click OK.
   The Fields dialog box closes and you return to the Properties window.

5. Back in the Properties window, set the PagedControlID to the name of the ListView control that you want paged, usually ListView1.

6. Set the PageSize property value to the number of items that should appear on the page at one time, such as 5.

When included in a <div> tag for style and formatting, the preceding steps create the following markup:

```xml
<div class="datapager">
    <asp:DataPager PageSize="5" ID="DataPager1" PagedControlID="ListView1" runat="server">
        <Fields><asp:NumericPagerField /></Fields>
    </asp:DataPager>
</div>
```

To position the DataPager, try this style:

```
```css
.datapager {
    clear: both; display: block; font-size: medium; text-align: center;
}
```

You can also put the DataPager control inside the ListView control’s required <LayoutTemplate> area. In that case, you don’t need to designate the PagedControlID property because the DataPager defaults to paging its container, which is the ListView.
This chapter gets you started with some common client-side behaviors and server-side image processing. The second half content is quite code intensive and somewhat advanced because I show how everything works in some detail. If you find the explanations heavy going, consider skipping them and just follow the steps without troubling yourself with what’s going on under the hood. You can always come back to pick up the gory details later.

Creating Rollover Effects

The rollover is probably the most common visual effect on Web pages, apart from those annoying animated advertisements. A rollover changes the appearance of a control according to what the mouse pointer is doing. A normal state shows the default appearance (the pointer is not on the control) and a hover state shows when the mouse pointer is over the control.

This section looks at creating rollovers that act on text and images.

Making a text rollover with a stylesheet

One of the easiest ways to create a rollover effect is to harness a built-in HTML style class. The anchor tag (<a>) has four pseudo-classes, :active, :hover, :link, and :visited. The :hover class applies to links with the
mouse hovering over them. You’ve probably seen styles like the following A selector that turns the hyperlink red when the mouse pointer passes over:

```
A:hover {color: red}
```

You can build on the pseudo-class by squeezing in a class selector:

```
A:button:hover {color: red}
```

This way, the special formatting happens when the anchor tag includes class="button":

```
<a class="button" href="nowhere.htm">This is my hyperlink</a>
```

The upshot is that you get a rollover without any programming (for free in geekspeak). What’s more, you can use the technique with the ASP.NET HyperLink and LinkButton controls because they render as anchors. Figure 14-1 shows a LinkButton control that resembles a button because its style rule defines borders, padding, and shading.

![Figure 14-1: A LinkButton with borders and shading.](image)

The following style rule gives Figure 14-1 its appearance. Notice that the rule includes a background image that it repeats horizontally.

```
A.button
{
    background-image: url(images/bbg.gif);
    background-repeat: repeat-x; background-color: #3399ff;
    border-color: #0f8bdc; border-style: solid; text-decoration: none;
    border-width: 1px; color: white; cursor: hand; font-size: 11px; margin: 4px;
    font-family: Verdana,Arial,Helvetica,sans-serif; padding: 3px 12px;
}
```

It takes only a couple of lines to create the distinctive hover appearance. The :hover pseudo-class references a light background image and makes the text black rather than white (see Figure 14-2):

```
A.button:hover
{
    background-image: url(images/bbgh.gif); color: black;
}
```
Here's the markup for a `LinkButton` that uses the `button` style class you see in Figure 14-2:

```xml
<asp:LinkButton ID="LinkButton1" runat="server" CssClass="button">Roll Over Me Now</asp:LinkButton>
```

The background bitmaps that add a gradient and flare to the link are slivers — only 1 pixel wide and 20 pixels high. By repeating, they effectively fill whatever width the text requires.

### Using JavaScript and images for rollovers

Web developers often use images as rollovers. The trick is to use JavaScript to detect when the mouse pointer passes over the image and then replace the normal image with an “over” image. Figure 14-3 shows an ASP.NET `Image` control and an `ImageButton` control. The control on the right is using the “over” (that is, brighter) version of the image because the mouse pointer is hovering. (There's an investigation to catch the joker who positioned the mouse pointer up the author's nose!)

Follow these steps to create rollover effects for the `Image` and `ImageButton` controls:
1. Find or create two images for the normal and “over” states and name the controls ken.gif and kenover.gif respectively. Put them in your project’s images folder.

Okay, it’s not mandatory to use my name for the images but if you don’t, remember to adjust the code for the image names you use.

2. From the Toolbox, drop an Image control and ImageButton control on an ASP.NET page.

3. Set the ImageUrl property for each control to ~/images/ken.gif (or whatever image name you used).

The tilde character (~) in the ImageUrl attribute is interpreted by ASP.NET as starting at the root of the Web. In this case, ASP.NET locates the image by going to the root and then back down into the images folder.

4. In Design view, double-click a blank area of the surface to create a handler for the Page Load event.

5. Use the following code inside the event handler subroutine:

   ```vbnet
   If Not IsPostBack Then
       Image1.Attributes.Add("onmouseover", _
           "this.src='images/kenover.gif'")
       Image1.Attributes.Add("onmouseout", _
           "this.src='images/ken.gif'")
       ImageButton1.Attributes.Add("onmouseover", _
           "this.src='images/kenover.gif'")
       ImageButton1.Attributes.Add("onmouseout", _
           "this.src='images/ken.gif'")
   End If
   ``

The preceding Visual Basic code adds onmouseover and onmouseout attributes to Image1 and ImageButton1 by using the Attributes collection’s Add() method. You can add any attribute to a server-side control this way. Here’s what the browser sees for Image1:

```html
<img id="Image1"
    onmouseover="this.src='images/kenover.gif'"
    onmouseout="this.src='images/ken.gif'"
    src="images/ken.gif" style="border-width:0px;" />
```

Here’s the rundown on the parts of the preceding HTML and JavaScript code that adjusts the image:

- **onmouseover**: The name of the event that you want to handle.
- **this**: A shortcut reference to the current object (<img...>) that’s hosting the script.
The `<img>` and `<input type="image">` both use the `src` attribute that points the way to the image file.

`images/ken.gif`: The location and name of the default image.

When the `onmouseover` event fires, the browser mumbles to itself, “Huh? Oh, the mouse just passed over that goofy picture. Let me see whether there’s a handler that wants to do anything about this situation. Yup, there’s a handler. Okay, some JavaScript wants the image control to render a different picture. And here’s the `src` value that describes where to get the picture!”

**Creating and Displaying Graphics on the Fly**

ASP.NET opens the world to many capabilities found in the .NET platform, such as GDI+ (GDI plus), Microsoft’s graphics technology. In this section, you create a composite graphic that displays user-submitted text.

The visible page, shown in Figure 14-4, accepts user input and displays the resulting bitmap in an ASP.NET `Image` control. Along the way, you use content caching and the ASP.NET `UpdatePanel` control to make the page more responsive.

![Goldie © 2007 Ken Cox](image)

**Figure 14-4:** Vertical text created on the fly.
Generating a custom image in ASP.NET

In this section, you create a very strange looking ASP.NET page that has no markup and no controls. Although you can browse to the page, you can’t view the HTML source because there isn’t any!

The sole purpose of generateimage.aspx is (can you guess from the name?) to read a static image, customize it, and send out the custom version. To create the page, follow these steps:

1. Add a single file ASP.NET page named generateimage.aspx to your project.
2. In Source view, remove all existing content from generateimage.aspx and replace it with the code in Listing 14-1.
3. Create or find a .jpg image that’s roughly 320 pixels high and 300 pixels wide, name it goldiewv.jpg, and put it in the images folder.

   In this example, I use a picture of Goldie at the Grand Canyon in Arizona. For those who love dogs, the picture is available from my Web site.


   You see the picture with today’s date in vertical, white text along the right side. Figure 14-5 shows the text portion of the picture.

5. With the browser still open, add the following query string to the end of generateimage.aspx so it looks like the following. The part you add is in bold:

   generateimage.aspx?text=My Query String

   When you view the page this time, the bitmap renders with the text “My Query String” instead of the date.
Listing 14-1:  generateimage.aspx Produces a .jpg Image

```vbnet
<%@ Page Language="VB" %>
<%@ OutputCache Duration="60" VaryByParam="text" %>
<%@ Import Namespace="System.Drawing" %>
<script runat="server">
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    Dim textstring As String
    If IsNothing(Request("text")) Or _
        Request("text") = "" Then
        textstring = Date.Now.ToLongDateString
    Else
        textstring = Left(Server.UrlDecode(Request("text")), 30)
    End If
    Dim grfont As New System.Drawing.Font("Tahoma", 17, _
        Drawing.FontStyle.Bold)
    Dim drawformat As New System.Drawing.StringFormat
    Dim goldieimage As Bitmap
    Try
        goldieimage = New Bitmap(Server.MapPath("~/images/goldievw.jpg"))
    Catch exc As Exception
        goldieimage = New Bitmap(300, 320)
    End Try
    Dim g = System.Drawing.Graphics.FromImage(goldieimage)
    g.SmoothingMode = Drawing.Drawing2D.SmoothingMode.HighSpeed
    drawformat.FormatFlags = Drawing.StringFormatFlags.DirectionVertical
    g.DrawString(textstring, grfont, _
        Drawing.Brushes.Black, 253, 3, drawformat)
    g.DrawString(textstring, grfont, _
        Drawing.Brushes.White, 250, 0, drawformat)
    Response.Clear()
    Response.ContentType = "image/jpeg"
    goldieimage.Save(Response.OutputStream, _
    goldieimage.Dispose()
    g.Dispose()
End Sub
</script>

Here's how Listing 14-1 breaks down:

1-12 The If...Else...End If sequence tests the built-in Request object to see whether a string of text (such as My Query String) was passed in as a parameter on a query string. A query string starts with a question mark (?) and includes one or more name/value pairs. In this case text is the name portion and My Query String is the value part.
If no text is available, the routine assigns today’s date as the default text. If there’s something usable in the query string, the routine captures the text — but not before reducing the text to 30 characters (maximum) with the Visual Basic `Left()` function.

Verify query strings before you use them in your code. Don’t assume they’re valid and friendly. As you discover in Chapter 17, all user input is evil until proven otherwise.

After you have an acceptable string to put on the bitmap, you can make choices about how the text should look. The code creates a `System.Drawing.Font` object, telling it to use Tahoma in bold with the size at 17 points.

The variable `goldieimage` holds a `System.Drawing.Bitmap` object that it attempts to create from a `.jpg` file.

The `Try...Catch...End Try` sequence copes with the possibility that the image `goldieww.jpg` can’t be found or the permissions don’t allow ASP.NET to read it. If there’s a problem, the Catch portion kicks in, and creates a bland image so the demonstration still works. For more on `Try` and `Catch`, see Chapter 21.

The variable `g` represents a `System.Drawing.Graphics` object. You can think of it as a workbench and canvas that helps you apply text, lines, colors, and shadings. The `FromImage` method accepts the `Bitmap` object to create a base image to build on.

The `DrawString()` method draws text over the image. You tell it what text to write, what font to use, the brush color, the starting coordinates, and the text direction (vertical in this case). Notice that the code calls `DrawString()` twice with different parameters. The first time it draws black text. The second time, it draws white text and offsets it three pixels to create a shadow effect.

Having constructed the image, the code needs to send it to the browser. The `Response` object is a type of shipping department within ASP.NET. It handles stuff that needs to go out to the browser such as cookies, HTML tags, headers, and more. The `Response` object’s `Clear()` method empties the Web server’s buffer in case there’s leftover content. Speaking of content, the `ContentType` property tells the browser what kind of thingy to expect. In this case, it’s a JPEG image that you identify with the standard MIME type, `image/jpeg`.

The `Bitmap` object’s `Save()` method is quite flexible. It can save the image to a disk drive or to a stream of bytes held in memory. In this case, the `Save()` method passes the stream of bytes to the `Response` object that pushes the data to the browser.
Updating and displaying the custom image

The user interface page accepts text input and renders the custom image. The odd thing is the way you refer to the location of the image. Instead of pointing the ASP.NET Image control to a static .jpg file, you point it to generateimage.aspx that you created previously. As you see in the code, you extend the URL of generateimage.aspx to pass along the text that you want to appear on the image. Follow these steps to create a page that updates and displays the custom image:

1. Add an AJAX Web Form page (yes, the AJAX version) called use image.aspx to your project (File › New File › AJAX Web Form).

2. In Design view, add an UpdatePanel control.

3. Drop a TextBox, Button, and Image control inside the UpdatePanel, as shown in Figure 14-6.

4. Double-click an empty area of the page to create a skeleton handler for the Page Load event.

5. In Source view, add the following code above the End Sub statement of the Page_Load subroutine:

   ```
   If Not IsPostBack Then
     Image1.ImageUrl = "generateimage.aspx"
   End If
   ```

   This code fetches a generated image the first time the page loads.

6. In Design view, double-click the Button control to create a handler for its Click event.
7. In Source view, insert the following code within the Button1_Click subroutine:

```csharp
Image1.ImageUrl = "generateimage.aspx?text=" & Server.UrlEncode(TextBox1.Text)
```

This code tells the Image control where to find the image but also builds a query string starting with the question mark (?). The Server object’s `UrlEncode()` method formats the text from the text box so that the text passes without difficulty on the query string.

The first time the page loads, the Image control looks to generateimage.aspx as the source of the image. Because generateimage.aspx doesn’t see anything on a query string, it creates the image with the date. If you type some text and click the button, the text you typed appears on the image.

For fun, try this: Put your cursor in the text box, hold down the Alt key and type 0169 on the keypad. Click the button to see a copyright symbol (©) on the image.

**Displaying Uploaded Image Files As Thumbnails**

Sharing files is one of the most popular pastimes on the Internet. Participants in a Web site upload pictures and videos for others to view. In this section, you create a small ASP.NET application that uploads a large image to the Web server and then immediately displays a thumbnail version of the image. In the following sections, you see that ASP.NET and the .NET Framework do most of the work. Your job is to assemble and organize the work crew.

**Accepting a file upload**

The goal in this section is to let a visitor upload an image file and store the image on the Web server. The code is picky about what it accepts: The file must have a .gif or .jpg extension. Anything else generates a warning.

Allowing uploads of any kind to a Web server requires you to give the user or the ASP.NET account `write` access to the file system on the Web server. Write access weakens security, so check with your administrator to be sure you’re not creating vulnerabilities.
To create a page that uploads images, follow these steps:

1. **Make sure that your project has an images folder.**

2. **Add an ASP.NET page named upld.aspx to your project (File ➪ New File ➪ Web Form).**

3. **From the Toolbox, drag and drop FileUpload, Button, Label, and ImageButton controls onto the page.**

   Figure 14-7 shows the resulting page in Design view.

4. **Select the ImageButton control, and in its Properties window (F4) set the Visible property to False.**

5. **Double-click the Button control to create a skeleton handler for its Click event.**

6. **In Source view, insert the following code in the Button1_Click subroutine:**

   ```vbnet
   If Not FileUpload1.HasFile Then
   Label1.Text = "You have not uploaded a file."
   ElseIf CheckExtension(FileUpload1.FileName) = False Then
   Label1.Text = "Only .gifs, and .jpgs, please."
   Else
   ImageButton1.Visible = True
   Try
   FileUpload1.SaveAs(Server.MapPath("~/images/" & FileUpload1.FileName)
   Label1.Text = "Thumbnail of: " & FileUpload1.FileName
   Catch exc As Exception
   Label1.Text = "There was a problem: " & exc.Message
   End Try
   End If
   ```
7. After the `End Sub` from the preceding routine, insert the following function:

```vbnet
Function CheckExtension(ByVal strFilename As String) As Boolean
    Dim strExt As String = _
        strFilename.Substring(strFilename.LastIndexOf(".")).ToLower()
    Return (strExt = ".jpg") Or (strExt = ".gif")
End Function
```

There’s a lot going on in the code in Step 6. The first statement checks the `FileUpload` control’s `HasFile` property to make sure that the user selected a file. If `HasFile` is `False`, assign some warning text to the `Label` control’s `Text` property to alert the user. The ` ElseIf` portion only executes if the user selected a file. The line’s role is to make sure that the user chose a `.gif` or `.jpg` extension for uploading.

**Checking the file extension**

To handle the extension validation, the main routine calls a helper function called `CheckExtension()`, passing the function the name of the file the user intends to upload. `CheckExtension()`, which you added to the page in Step 7, pulls some strings. Using the Visual Basic `Substring()`, `LastIndexOf()`, and `ToLower()` functions, the helper collects the characters starting with the last period (.) to the end of the filename, and converts those letters to lowercase. The `Return` line uses `Boolean` logic (see the sidebar “Boolean logic: It’s True and not False”) to determine whether the file extension is `.jpg` or `.gif` and reports `True` or `False` as to whether all is well.

The main routine examines the `CheckExtension()` verdict on the filename. If the report is `False` (an allowed extension wasn’t found), the routine reports the problem and the `If...End If` routine is done.

** Catching a bad `SaveAs`**

Assuming the file extension checked out okay, the `Else` section executes. The first step is to make the `ImageButton` visible so it can display the thumbnail image whenever it arrives. Because anything to do with saving files is fraught with danger in ASP.NET, wrap the remaining statements in a `Try...Catch...End Try` sequence. The `FileUpload1.SaveAs()` method tries to save the file to the `images` directory on the server by using the file’s name. To let the user know what’s going on, the `Label` control is told to display the name of the file.

**Transmitting info on a query string**

Here’s the best part: The code instructs the `ImageButton` to look for the thumbnail image from a `handler` called `thumbnailer.ashx`. An ASP.NET handler is a special-purpose file that toils without a visible interface. It’s motto is “No HTML to see here folks. Move along.” You find out more about `thumbnailer.ashx` in the next section, but suffice to say that the query string transmits the name of the thumbnail image that the `ImageButton` wants to display.
Boolean logic: It’s True and not False

To a geek, the common phrase “Two wrongs don’t make a right” rings true because it resembles Boolean logic. Look at the last line of code from the CheckExtension() function (used in the earlier section, “Accepting a file upload”). Say the value of strExt is .exe.

    Return (strExt = ".jpg") Or (strExt = ".gif")

Boolean logic starts on the innermost brackets and works to the outside. Here’s how .NET’s inner voice talks its way through the process:

Is strExt equal to .jpg? No, so that result is False.

Is strExt equal to .gif? No, so this result is also False.

Now I have False on the left, the logical operator Or in the middle, and False on the right. Look this up in the Or Truth table.

<table>
<thead>
<tr>
<th>First Expression Is...</th>
<th>Second Expression Is...</th>
<th>Result of Or Is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
</tbody>
</table>

Hmmmm... False Or False results in False, so that’s what I’ll return.

Now, say the value of strExt is .gif. Listen while .NET does its monologue:

Is strExt equal to .jpg? Nope. That’s False.

Is strExt equal to .gif? Yes! We have a match! That result is True.

This time, I have a False on the left, the logical operator Or in the middle, and True on the right. Checking the Or Truth table, I see that I’m required to return True for that combination.

Here’s one more example of Boolean logic. This time, it’s a test for whether the user has uploaded a file. Say the user uploaded something, making the HasFile property value True:

    If Not FileUpload1.HasFile Then

Here’s .NET’s analysis of the preceding line:

The HasFile property is True. That was easy.

There’s a logical Not in front of it. Check the truth table that rules on the Not operator.

<table>
<thead>
<tr>
<th>The Expression Is...</th>
<th>Result of Not Is...</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

(continued)
Getting the real file path

That’s almost all of note in the routine, except for the Server object’s MapPath() function. MapPath() looks at a Web directory name and tells ASP.NET exactly where to find the directory on the Web server’s file system. As usual, if something goes wrong, the Catch part reports the bad news on the Label control.

Creating a thumbnail image WebHandler

In the earlier section, “Generating a custom image in ASP.NET,” you use a HTML-less ASP.NET page to generate an image. Here, you use an ASP.NET WebHandler to produce a thumbnail version of an image. ASP.NET recognizes a handler with the .ashx extension as a workhorse. A WebHandler has a different makeup than an .aspx file, including mandatory content.

To create a thumbnail image WebHandler, follow these steps:

1. Create or locate a .jpg image (about 250 pixels by 250 pixels) to use as a generic “not found” image. Put the file in your project’s images folder with the name notfound.jpg.

2. Add a WebHandler file named thumbnailer.ashx to your project (File⇒New File⇒Generic Handler).
Chapter 14: Dynamic Effects, Images, and Rollovers
The skeleton WebHandler code includes a subroutine called Process
Request() and a property called IsReusable() that must remain,
even though you don’t do anything with IsReusable() in this
procedure.
3. In the WebHandler’s Imports section, add the following imports:
Imports System.Drawing
Imports System.Drawing.Imaging

4. Replace the entire ProcessRequest() subroutine with Listing 14-2.
This is the core code that reads an existing image and creates a
thumbnail copy.
5. Browse to thumbnailer.ashx to view the result.
The browser shows a thumbnail version of the “not found” image.

Listing 14-2:

Creating a Thumbnail Image in thumbnailer.ashx.

Public Sub ProcessRequest(ByVal context As HttpContext) _
Implements IHttpHandler.ProcessRequest
Dim imgpath As String = “”
Dim callback As Image.GetThumbnailImageAbort = Nothing
Dim callbackData As IntPtr
Dim thumbnailimg As Image
Dim largeimg As Image = Nothing
If Not IsNothing(context.Request(“file”)) Then
imgpath = context.Server.UrlDecode(context.Request(“file”))
End If
Try
Catch exc As Exception
End Try
thumbnailimg =largeimg.GetThumbnailImage(40,50,callback,callbackData)
context.Response.ContentType = “image/Jpeg”
End Sub

➝8
➝10
➝11
➝12
➝14

Here’s how Listing 14-2 works:

➝8-10 It starts by looking for a query string variable called file whose
value is the name of the image to use as the basis for the thumbnail.

➝11

The Try keyword indicates some risky territory ahead where the
Server.MapPath() method returns a file system path (including
the filename) and the Image object’s FromFile() method tries to
read an image from that path.

➝12-14 If the image isn’t there, Image.FromFile() throws an exception.
As a backup plan, the Catch clause fetches notfound.jpg, the
default image you added at the start of the preceding steps.

237


It’s remarkable that you generate the thumbnail image with one line of code. The `Image` object’s `GetThumbnailImage()` method scales down the source image based on the two size parameters (width and height). ASP.NET doesn’t use the third and fourth parameters (`callback` and `callbackData`), but you must provide them to keep the `Image` object happy.

After the thumbnail image is ready, tell the calling routine the `ContentType` and use the `Save()` method to ship out the bytes. This is the same as in the previous section, “Generating a custom image in ASP.NET.”

### Displaying an uploaded image as a thumbnail

The page is now in place and the WebHandler is ready to handle a task. When you run `upld.aspx`, browse to a large `.jpg` image on your local machine and click Upload. The routine uploads the file. The `ImageButton` control asks `thumbnailer.ashx` to create a thumbnail version for display. Figure 14-8 shows the result in the browser, including the name of the original image and the tiny thumbnail version.

An interesting enhancement would be to use the `ImageButton`’s `Click` event to browse to the original image, full-size.
Chapter 15

Enhancing Pages with the AJAX Control Toolkit

In This Chapter

- Completing data as users type
- Using a lookup Web service
- Masking and watermarking text boxes
- Creating a pop-up calendar
- Keeping content on top

Microsoft is often late to technology parties, but the “embrace and extend” philosophy pays off when the Redmond giant gets rolling. In the case at hand, Microsoft introduced ASP.NET AJAX as a framework for creating dynamic, interactive pages. Suddenly millions of geeks have another platform to play on and show off their stuff.

This chapter looks at a handful of free add-ons that make ASP.NET pages easier to use and more interesting. The basis for the chapter, the AJAX Control Toolkit, shows skilled people grasping a new technology and producing results that the original designers never dreamed of.

Introducing the AJAX Control Toolkit

The AJAX Control Toolkit is a set of AJAX enhancements to existing ASP.NET controls. The project’s goal is “to be the biggest and best collection of web-client components available.” A group of developers within Microsoft started the toolkit and then opened it to volunteer contributors.

See Chapter 2 for instructions on downloading and installing the AJAX Control Toolkit.
AJAX extenders add dynamic capabilities to Microsoft’s standard ASP.NET controls. For example, Figure 15-1 shows the toolkit’s ValidatorCallout control. The ValidatorCallout control changes the behavior of ASP.NET validation controls (such as the RangeValidator), so the error message appears as a floating callout box rather than as flat text on the page. You can also include a custom icon, as you see in Figure 15-1.

An extender also makes changes to the affected control’s Properties window by adding properties. As shown in Figure 15-2, an Extenders category is at the top, and within it, the list of extenders. (Figure 15-2 has only one extender, RangeValidator1_ValidatorCalloutExtender.)
Under each extender’s node, you find the added properties that you can con-
figure. In this case, the WarningIconImageUrl property points to a custom
image. If you view the Properties window in alphabetical order (my prefer-
ence), it’s harder to find the node containing the extended properties. By
default, the extender’s ID starts with the name of the control it’s extending, so
you can use that as a clue.

**Automatically Completing Data As the User Types**

The AutoCompleteExtender enhances the ASP.NET TextBox control. As
the user types characters, the extender monitors the input and tries to match
words or data provided by a Web service. (For details on working with Web
services, see Chapter 9.)

In this section, you create a text box that helps users tag a blog post. For blog
readers, the tags are often displayed in a cloud where the size and font of the
tags indicate the most popular subjects.

**Preparing the word list**

The Web service reads from a list of words that you create as a text file. (You
could get the words from a database or XML file, but a flat text file is less
complicated.) Follow these steps to create the list of tag words:

1. Add a text file named cloudtags.txt to your App_Data folder (select
   App_Data; then choose File ➤ New File ➤ Text File ➤ Add).
2. In cloudtags.txt, add the tags, one name per line followed by a
carriage return.

Using the following example tags (from my ASP.NET blog at
http://weblogs.asp.net/kencox/) will make it easier to follow
along later. At a minimum, enter the names that start with V:

- ASP.NET
- Visual Studio
- AJAX
- C#
- DataContext
- Linq to SQL
- Vista
- .NET
- Atlas
- Community News
- IIS
- ListView

(continued)
Creating the data lookup Web service

While the user types, the AutoCompleteExtender (that you add in the next section) asks the Web service to return words or names that begin with the letters entered so far into your tag list. Follow these steps to create the data lookup Web service:

1. Add a Web Service called cloud.asmx to your project (File ➤ New File ➤ Web Service ➤ Add).
2. Add the complete contents of the Web service (see Listing 15-1).
3. Browse to cloud.asmx and click the CloudList link to reach the Web service test page.
4. Type vi as the prefixText and 5 as the count and then click Invoke.

The result of the test is an array with three strings:

<string>Vista</string>
<string>Visual Basic</string>
<string>Visual Studio</string>

Listing 15-1: Data Lookup Web Service

```vbnet
<%@ WebService Language="VB" Class="cloud" %>
<system.web.script.services.scriptservice()> _  
<webservice(namespace:="http://kencox.ca/*")> _  
<webservicebinding(conformsto:="WsIProfiles.BasicProfile1_1")> _
Public Class cloud
Inherits System.web.Services.WebService
```
Public Function CloudList(ByVal prefixText As String, ByVal count As Integer) As String()
    Dim autoCompleteWordList As String()
    autoCompleteWordList = CType(Context.Cache("autoCompleteWordList"), String())
    If autoCompleteWordList Is Nothing Then
        autoCompleteWordList = System.IO.File.ReadAllLines(Server.MapPath("~/App_Data/cloudtags.txt"))
    End If
    Dim strquery = From wrd In autoCompleteWordList
        Where wrd.StartsWith(prefixText, StringComparison.CurrentCultureIgnoreCase)
        Take (count)
        Order By wrd
    Return strquery.ToArray()
End Function

Briefly, here’s how Listing 15-1 works:

→18-22 It fetches the list of words from cloudtags.txt and inserts the array into the cache.

→23-27 A LINQ query looks through the array for entries that start with the text that was passed in. It uses LINQ’s Take() method to get as many items as the calling routine requested in the count variable.

→28 The ToArray() method stuffs the query results into an array and that’s what’s returned in the SOAP envelope to the calling control.

For more detail on the query, see Chapter 7. To understand the syntax in the Web service, refer to Chapter 9 and to the cheat sheet inside the front cover.

Creating the data lookup page

With the Web service in place, you have a source for the lookup names. To create a page that consumes the data, follow these steps:

1. Add an AJAX Web form page (hey, you’re using AJAX here!) called tagit.aspx to your project. (Choose the project name: File: New File. Select AJAX Web Form and click Add.)

2. In Design view, add an ASP.NET TextBox control to the page.

3. From the TextBox control’s Tasks menu, choose Add Extender.

   If there’s no Add Extender item, make sure you’ve installed the AJAX Control Toolkit, as described in Chapter 2.
4. In the Extender Wizard dialog box, select the AutoCompleteExtender icon.

5. Accept the default ID for the extender (probably TextBox1_AutoCompleteExtender) and click OK.

6. Open the Properties window for TextBox1 (F4) and within the TextBox1_AutoCompleteExtender node, set the following properties to the corresponding values:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DelimiterCharacters</td>
<td>; , : (that's a semi-colon, comma, space, and colon)</td>
</tr>
<tr>
<td>MinimumPrefixLength</td>
<td>1</td>
</tr>
<tr>
<td>ServiceMethod</td>
<td>CloudList</td>
</tr>
<tr>
<td>ServicePath</td>
<td>cloud.asmx</td>
</tr>
</tbody>
</table>

When you browse to tagit.aspx, start typing the letter v in the text box. As shown in Figure 15-3, typing the letter v produces a sorted drop-down list of the items that start with that letter. When you type more letters, the matches narrow.

If you have a very large number of completion words, set the MinimumPrefixLength value higher to reduce the number of initial matches.

Helping Users Understand What to Enter

Web forms with many labels and input areas become busy and overwhelming. You can reduce the clutter by removing labels such as First Name: and putting First Name Goes Here as a text box prompt (Microsoft calls it a watermark), as shown in Figure 15-4.
The advantage to using the TextBoxWatermarkExtender is that the pre-filled prompt text disappears when the text box becomes active (gets the focus in geekspeak).

Enhancing a text box with the TextBoxWatermarkExtender

As with other extenders, the TextBoxWatermarkExtender enhances a built-in control. In this case, it’s a TextBox. Follow these steps to implement a watermark on a text box:

1. Add an ASP.NET TextBox control to an AJAX Web form page.
2. From the TextBox control’s Tasks menu, choose Add Extender.
3. In the Choose an Extender dialog box, select the TextBoxWatermarkExtender icon and then click OK.
4. Open the TextBox control’s Properties window (F4) and locate the extender’s node (probably TextBox1_TextBoxWatermarkExtender).
5. Set the WatermarkText property to First Name Goes Here.

At runtime, the prompt text appears and then disappears when you start typing in the text box.

Adding style to a watermark

To avoid confusion, it helps if the watermark text is dramatically different from the text the user enters. You can create a different appearance by applying a different style for the watermark. Follow these steps to create and apply a distinct watermark style:
1. In Source view, add the following style markup in the `<head></head>` section of the page:

```html
<style type="text/css">
  .watermarkstyle
  {color: white;background-color: gray;}
</style>
```

2. In Design view, select the TextBox control, open its Properties window (F4), and in the extender node (probably `TextBox1_WatermarkExtender`), set the `WatermarkCssClass` property to `watermarkstyle`.

When you browse to the page, the watermark uses white text on a gray background. As soon as you start typing, the text box assumes its normal appearance.

**Guiding Input with a Masked Text Box**

As you discover in Chapter 19, it’s dangerous to trust users to enter valid data. Even if their input isn’t evil (although you should assume it is until proven otherwise!), whatever users type can be light years from what you intend. If you ask them *not* to enter dashes, expect dashes anyway.

Formatted input with a mask lets you guide users whenever they enter a date, IP address, telephone number, currency value, or credit card number. For example, Figure 15-5 shows the use of a masked text box for a North American telephone number format. The text box at the top is complete and the mask elements have disappeared. The lower text box shows a formatted number with the cursor still in the text box. The commonly used telephone number format puts the area code (such as 705) in parentheses, followed by a space, and then three digits, a dash, and the remaining four digits.

![Figure 15-5: A mask for a North American telephone format.](image)
Creating a masked input

Follow these steps to create the masked input used in Figure 15-5:

1. Add an ASP.NET TextBox control to an AJAX Web form.
2. From the TextBox control's Tasks menu, choose Add Extender.
3. In the Choose an Extender dialog box, select the MaskedEditExtender and then click OK.
4. In the TextBox control's Properties window (F4), in the MaskedEditExtender node (usually TextBox1_MaskedEditExtender), set the Mask property to (999) 999-9999 and the MaskType property to Number.

In the browser, the enhanced text box shows the brackets and dash. It allows only digits in the spaces.

Using masks and custom characters

Masked edit controls use designated characters as placeholders for the mask. In the preceding example, the character 9 tells the text box to accept only a number in that position. The brackets ( and ) represent themselves. Here are the mask characters for the MaskedEditExtender control:

- 9: Must be a numeric character, 0–9
- L: Only a letter of the alphabet is allowed
- $: Must be a letter of the alphabet or a space
- C: Accepts only a custom character (case sensitive)
- A: Must be a letter of the alphabet or a custom character
- N: Accepts a number or custom character
- ?: Any character is fine, thanks

A custom character is a character that you specifically want to allow in a C, A, or N position. You tell the control which characters to accept by assigning the character (or characters) as the value of the Filtered property. For example, say you want to allow users to enter certain currency symbols, such as the euro in Figure 15-6. Instead of resorting to the generic ? (any character) option, you can restrict input to the euro, pound, yen, or dollar sign by defining them as custom characters.
To use custom characters in a mask, as in Figure 15-6, follow these steps:

1. **In Design view, add an ASP.NET TextBox control to an AJAX Web form.**

2. **From the Toolbox, drag and drop the MaskedEditExtender control onto the TextBox control.**
   
   This step shows you an alternate design technique — dropping an extender directly onto the control that it extends.

3. **Open the TextBox control’s Properties window (F4) and navigate to the extender node (usually TextBox1_MaskedEditExtender).**

4. **Set the extender’s properties to the values shown in the following table:**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filtered</td>
<td>€£¥$ (that string is a euro sign, pound sign, yen sign, and dollar sign).</td>
</tr>
<tr>
<td>Mask</td>
<td>C99999.99 (the C stands for custom character)</td>
</tr>
<tr>
<td>MaskType</td>
<td>Number</td>
</tr>
<tr>
<td>PromptCharacter</td>
<td>- (a dash)</td>
</tr>
</tbody>
</table>

   If you need help with generating the characters, see the sidebar, “Entering custom characters.”

The preceding steps generate the following markup. You can see the Filtered property value in bold:

```xml
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<cc1:MaskedEditExtender ID="TextBox1_MaskedEditExtender" runat="server" Filtered="€£¥$" Mask="C99999.99" MaskType="Number" PromptCharacter="-" TargetControlID="TextBox1">
</cc1:MaskedEditExtender>
```

When you browse to the page, you can use Alt-0128 to enter the euro sign in the custom character location, indicated by the C in the Mask attribute’s value.
Choosing Dates with a Calendar

Entering dates is a risky business because there are so many formats. Does 08/06/52 mean August 6, 1952 or June 8, 1952? The better way is to let users pick the date from a calendar.

The AJAX Control Toolkit CalendarExtender control enhances the capabilities of the ASP.NET TextBox control. By default, the calendar appears when the user puts the pointer in the text box. If you’re concerned that users won’t find this behavior intuitive, you can configure the control so users click an icon (such as a calendar image) to invoke the calendar.

To create a date-picker with the AJAX Control Toolkit CalendarExtender control, follow these steps.

1. **Add a TextBox control to an AJAX Web form.**
   Whenever you use ASP.NET AJAX controls, you must include the ScriptManager control as the first control on the page. The AJAX Web form template adds the ScriptManager for you.

2. **Add an ASP.NET Image control to the form and set its ImageUrl property to the location of a calendar icon or other graphic that suggests date input.**
   The image acts as a button at runtime, but don’t use an ImageButton control. The extender takes care of the click action.

3. **Add the following style class inside the <head></head> section of the page. Add <style></style> tags if you need them:**

---

**Entering custom characters**

Keyboards built for use in North America may not have euro, pound, or yen keys. To use these and other special characters, you need to know the character’s value. When you have the value, you can insert the character in your page by using an Alt+ sequence. That’s where you hold down the Alt key and type the four numbers of the character’s value on the numeric keypad.

For example, to generate the euro sign on an older North American keyboard in Windows, hold down the Alt key, type 0128 and release the Alt key. (Use all four digits, even when the first is a zero.)

Here are the codes for the currency symbols used in this section:

- **euro:** Alt+0128
- **pound:** Alt+0163
- **yen:** Alt+0165

Many Web sites include charts of character codes. Windows includes a utility called Character Map that lets you pick symbols and read the corresponding keystroke. Look for Character Map by choosing Accessories ➪ System Tools.

Here are the codes for the currency symbols used in this section:

- **euro:** Alt+0128
- **pound:** Alt+0163
- **yen:** Alt+0165

Many Web sites include charts of character codes. Windows includes a utility called Character Map that lets you pick symbols and read the corresponding keystroke. Look for Character Map by choosing Accessories ➪ System Tools.
The CalendarExtender control has built-in styles for the headers, months, and days with style names, such as `ajax__calendar_container`. You can override the default styles by providing your own style class. The preceding style rule changes the background color.

4. Click the TextBox control’s Smart Tag and then select Add Extender.
5. Click the icon for the CalendarExtender control and then click Add.
6. Using the TextBox control’s Smart Tag, add another extender, the TextBoxWatermarkExtender.
   This step demonstrates that you can have more than one extender per control.
7. Open the TextBox control’s Properties window and set the following properties and values under the CalendarExtender node (usually TextBox1_CalendarExtender):

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CssClass</td>
<td>PopupUpCalendar</td>
</tr>
<tr>
<td>Format</td>
<td>MMMM d, yyyy</td>
</tr>
<tr>
<td>PopupButtonID</td>
<td>Image1 (or whatever the ID is of the calendar icon image you used)</td>
</tr>
</tbody>
</table>

8. In the TextBoxWatermarkExtender’s node (usually TextBox1_ TextBoxWatermarkExtender), set the WatermarkText property value to the text Click the calendar icon.

When you browse to the page and click the date icon, the calendar appears, as shown in Figure 15-7. Click the date, and the calendar disappears leaving the formatted date in the text box, as shown in Figure 15-8.
The calendar inserts a date, but the user can type anything else in the text box. Make sure you continue to validate all user input as described in Chapter 19.

Positioning Content to Stay on Top

The AlwaysVisibleControlExtender lets you float text over the page even when the user scrolls in the browser.

Use this control sparingly. I’d never suggest this technique for annoying advertisements, but it could be handy to park a helpful notice on the screen and update it as required.

In this section, you create the page shown in Figure 15-9. It’s hard to tell from the figure, but when you scroll down the page, the advertisement informational panel stays in place.
Creating a floating style

The panel floats above the text because it uses absolute positioning within the style. In this section, you create two styles for the floating content. The first style, alwaysvisible, defines the size (in pixels) of the outer container, a dotted red border, and a light gray background.

As a recommended option, include style rules that establish a fixed position such as 350 pixels from the left side of the browser pane and 10 pixels down from the top of the pane. Providing these in the style rule helps as you design the page and avoids a screen flash at runtime.

The second style, innercontent, formats the text. Its rules include the alignment, font, size, and color. The inner container stretches to 100 percent of its parent’s width and height.

Follow these steps to add the style for the floating content:

1. Add an AJAX Web form named alwaysvisible.aspx to your project (File ➤ New File ➤ AJAX Web Form ➤ Add).
2. In Source view, insert the following style markup within <style> </style> tags:

```html
.alwaysvisible
{
    position: fixed; left: 350px; top: 10px; background-color: gainsboro;
    height: 100px; width: 200px; border: dotted 4px red;
}
.innercontent
{
    vertical-align: middle; width: 100%; height: 100%; text-align: center;
    font-family: Comic Sans MS; font-size: x-large;
    color: black; padding: 10px; text-decoration: none;
}
```

Adding Panel controls to make <div>s

It’s hard to know where to position the floating content until you see the control in the designer. You may need to adjust the top and left values after working with the Panel controls.

You create floating content with HTML <div> controls and position attributes in style sheet rules. The ASP.NET Panel control renders as a <div>, making it a handy container. Follow these steps to create the Panel controls and other markup:
1. From the Toolbox, add an ASP.NET Panel control to the page.

2. In the Panel control's Properties window (F4), set the CssClass property to alwaysvisible.

   By creating the style first, you give the Panel its shape at design-time, making the remaining steps easier.

3. Drag a second Panel control, and drop it inside the existing Panel.

4. Set the second Panel control's CssClass property to innercontent.

5. Drag an ASP.NET HyperLink control to the panel and set its NavigateUrl property to http://www.kencox.ca and its Text property to Buy My Book!.

   The URL to my Web site and the phrase Buy My Book are important to the success of this step. Please type carefully. (Just kidding!)

6. Add an ASP.NET Image control to the panel and set the ImageUrl property to the location of a small image (for example, http://www.kencox.ca/images/dummyguy.gif).

That completes the design of the floating content. In the next section, you add the AlwaysVisibleControlExtender to ensure that the Panel remains visible even when scrolling on a long page.

### Adding the AlwaysVisibleControlExtender on a page

Like all controls from the AJAX Control Toolkit, the AlwaysVisibleControlExtender requires the JavaScript libraries provided by ASP.NET AJAX. A ScriptManager control ensures that the required libraries are available. Follow these steps to add the AlwaysVisibleControlExtender and configure it:

1. Select the first Panel control (probably Panel1) that you added to the page and click the Smart Tag to open the Tasks menu.

   Selecting a Panel can be difficult in the Designer. It's much faster to switch to Source view, put the cursor on the panel's ID, and switch back to Design view.

2. From the Panel control's Tasks menu, click Add Extender, select the AlwaysVisibleControlExtender, and click OK.

3. Open the Properties window (F4) for the Panel control that you're extending and expand the extender's node (usually Panel1_AlwaysVisibleControlExtender).
4. Set the HorizontalOffset property to 350 and the VerticalOffset property to 10.

These are the same values (in pixels) that you used in the .always visible style sheet rule.

The page is ready to view. The only problem is that you need enough content to force scrolling. You can start typing your life story or just copy and paste content from my blog (http://weblogs.asp.net/kencox/).
The rich Internet application (RIA) is the next phase in the evolution of Web sites where interactive, full-motion video clips replace text and static content. There’s competition to sell tools for building these apps. For example, Microsoft introduced the Silverlight browser plug-in and supporting development environments to give creators an alternative to the ubiquitous Adobe Flash technology and a reason to buy new design software.

One aspect of rich media is the high-fidelity, faithful rendering and printing of the original content. To that end, this chapter shows how ASP.NET can deliver PDF, Word, and Excel files that the end user views in the designated programs.

If you’re intending to create elaborate Silverlight content, you probably want to use dedicated design software, such as Microsoft Expression Blend. Because this book’s about ASP.NET and not about Blend, this chapter focuses on hosting Silverlight in a Web page.

Creating Your First Rays of Silverlight

Silverlight is a cross-browser, cross-platform plug-in for putting rich, interactive media into Web pages. The plug-in’s basic content is in the XAML markup language, an XML format that describes what to display. You create the trivial Silverlight application shown in Figure 16-1 with the Visual Web Developer text editor.
Almost everything that you need to get started with Silverlight in ASP.NET is available in a free download from Microsoft’s Web site. Rather than advising you to type a ridiculous-looking (and possibly outdated) URL such as

After downloading the .msi file, double-click the filename and follow the installer’s prompts. If it offers to install templates for Visual Studio 2005, click Skip.

**Setting up the Web project**

Silverlight applications run inside a browser page and call on external JavaScript libraries. Follow these steps to set up the ASP.NET project to host a Silverlight plug-in.

1. **In Visual Web Developer, create a new Visual Basic file system Web site named** `hostsilverlight` *(File ➪ New Web Site ➪ ASP.NET Web Site ➪ c:\hostsilverlight ➪ OK)*.

2. **Using Windows Explorer, locate** `Silverlight.js` **in your file system.**

   By default, the file is in `C:\Program Files\Microsoft Silverlight 1.0 SDK\Tools\Silverlight.js`.

   If you’re creating a non-English page, use one of the localized versions in the localized folder and rename the file to `Silverlight.js`.

3. **Drag `Silverlight.js` from Windows Explorer and drop it onto the project name in Solution Explorer.**

4. **Add a new JavaScript file named** `CreateSilverlight.js` **to your project** *(File ➪ New File ➪ Jscript File ➪ Add)* and use the following at the complete contents of `CreateSilverlight.js`:
function createSilverlight() {
    Silverlight.createObject(
        "source.xaml", parentElement,"mySilverlightPlugin",
        {
            width:'300', height:'300', inplaceInstallPrompt:false,
            background:'white', isWindowless:'false', framerate:'24',
            version:'1.0'
        },
        {
            onError:null, onLoad:null
        },
        null);
}

The preceding script creates the initial structure for the Silverlight control on the Web page and points to the content file, source.xaml. The script sets up an object and configures its height, width, and background color.

JavaScript is case-sensitive. If you encounter errors, check whether you used the wrong case.

5. Open an ASP.NET page in Source view and above the closing </form> tag, add the following markup:

```html
<div id="sldiv"></div>
```

The preceding creates and identifies a container tag for the Silverlight plug-in. Make sure that you use the same ID value here as in Step 7.

6. Still in Source view, in the line above the closing </head> tag, add the following script:

```html
<script type="text/javascript" src="Silverlight.js"></script>
<script type="text/javascript" src="createSilverlight.js"></script>
```

The preceding JavaScript imports the JavaScript files that the page needs to put the Silverlight plug-in on the page.

7. At the bottom of the source code, just before the closing </body> tag, add the following JavaScript:

```html
<script type="text/javascript">
    var parentElement =
        document.getElementById("sldiv");
    createSilverlight();
</script>
```

The preceding script kicks off the code that puts a Silverlight control on the page. You put this part at the end of the HTML so that the browser has the markup in place before the script tries to locate a container object (namely, sldiv) or other JavaScript routines.
You probably noticed that you’re missing one important piece: `source.xaml`. You can give Silverlight some content in the next section.

### Creating static XAML content

In this chapter, you’re using the slow, low-tech, bare-hands approach to creating this Silverlight application. The advantage is that you find out a little about what goes into a XAML file in case you ever need to tweak something that a tool has generated. Follow these steps to create a XAML canvas and include some content:

1. **In your VWD project, add a text file named** `source.xaml` *(File ➤ New File ➤ Text File ➤ Add).*

2. **Add the following markup to** `source.xaml`:

    ```xml
    <Canvas xmlns="http://schemas.microsoft.com/client/2007">
        <Ellipse Height="200" Width="200"
            Canvas.Left="0" Canvas.Top="0"
            Stroke="DarkGray" StrokeThickness="20"
            Fill="lightgray"/>
    </Canvas>
    ```

    The preceding establishes a `<Canvas>` object as the root element for the file and the container for the `<Ellipse>` shape that follows. Because the height and width of the ellipse are equal, the result is a circle. The `Canvas.Left` and `Canvas.Top` attributes position the circle at the top left of its container. The `Stroke` and `StrokeThickness` attributes create the circle’s border, and `Fill` paints the inside of the circle light gray.

You can run the ASP.NET page to view the Silverlight circle. However, expect a few hurdles before you see the circle in your browser. If the Silverlight plug-in hasn’t yet been installed, you need to click the logo (shown in Figure 16-2) to start the download and installation process. Along the way, you need to accept all sorts of dire warnings and allow ActiveX controls to run in the browser. In Windows Vista, you need to show your passport at several checkpoints and confirm ten times in writing that it’s really, really okay for this to happen.
Just so the Silverlight canvas isn’t too bare and monochromatic, Listing 16-1 provides more markup that completes the scene in Figure 16-1. This code needs to go before the closing </Canvas> tag in source.xaml.

**Listing 16-1: Adding a Color Gradient to XAML**

```xml
<Path Fill="Gray"
    Data="M 1,50 C 10,300 300,-200 300,100" />

<Rectangle Width="140" Height="100"
    Canvas.Left="55" Canvas.Top="120">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Color="Yellow" Offset="0.0" />
            <GradientStop Color="Red" Offset="0.50" />
            <GradientStop Color="White" Offset="1.0" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

Here’s what’s happening in Listing 16-1:

1-2 The first tag, `<Path>`, creates a shape according to the mini-language that’s passed as the value of the `Data` attribute. The `M 1,50` tells Silverlight to create a `startPoint` at position 1, 50 and start drawing (moving) to the next location. The `C` creates a curve according to the gridlike control points that follow. You can find a lot more about drawing lines (vertical and horizontal) in the Silverlight SDK documentation. Look for “Path Markup Syntax” in the index.

4 The `<Rectangle>` tag not only positions the rectangle object but it includes instructions on how to fill the interior with gradient colors.

9-11 The `<GradientStop>` tag includes an `Offset` attribute whose value indicates the color transition point.

**Embedding Silverlight with the ASP.NET Silverlight Control**

The ASP.NET 3.5 Extensions are controls that Microsoft added after the initial ASP.NET 3.5 release. This section discusses the ASP.NET Silverlight control that was available as a preview at the time of writing. Keep in mind that the ASP.NET team may change property names and other details in subsequent releases. Also, check occasionally at the official ASP.NET site at www.asp.net for a release version.
To download and install the ASP.NET Futures, follow these steps:

1. **Locate the latest version of the Extensions by browsing to**
   www.microsoft.com/downloads **and searching for ASP.NET 3.5 Extensions.**
2. **Download the installer file to a temporary directory.**
3. **In Windows Explorer, double-click the filename and follow the installation prompts and security warnings.**
4. **After the installation finishes, create an ASP.NET 3.5 Extensions Web Site in**
   c:\aspxnet35ext\ (File: New Web Site: ASP.NET 3.5 Extensions Web Site).
5. **Open an ASP.NET page in Visual Web Developer and check whether the controls appear in a new ASP.NET 3.5 Extensions category in the Toolbox.**

   If you don’t find the added controls, check the FAQ section at www.kencox.ca for the latest instructions on fixing up the Toolbox.

---

**Hosting Silverlight with the ASP.NET Silverlight control**

The Silverlight control automates most of the steps used in the section “Creating Your First Rays of Silverlight,” earlier in this chapter. Follow these steps to build a page that hosts the Silverlight plugin:

1. **Add an ASP.NET AJAX Web form named slxaml.aspx to the project you created in the preceding section.**
   
   You need the AJAX version of the Web form because that template installs the ScriptManager control required by the ASP.NET 3.5 Silverlight control.

2. **Add a text file named textxaml.xaml to your project (File: New File: Text File: Add).**

3. **Add the following markup to textxaml.xaml:**

   ```xml
   <Canvas
   <TextBlock Canvas.Left="0" Canvas.Top="0"
     FontFamily="Comic Sans MS" FontSize="25">
     http://www.kencox.ca/
   </TextBlock.Foreground>
   <LinearGradientBrush StartPoint="0,0" EndPoint="1,1">
     <GradientStop Color="Red" Offset="0.0" />
     <GradientStop Color="Orange" Offset="0.2" />
     <GradientStop Color="Yellow" Offset="0.4" />
     <GradientStop Color="Green" Offset="0.6" />
   </LinearGradientBrush>
   ```
To create a drop shadow effect with text, use an additional `<TextBlock>` element with the same content but offset it by a few pixels:

```xml
<TextBlock Canvas.Left="2" Canvas.Top="2"
    FontFamily="Comic Sans MS" FontSize="25">
    http://www.kencox.ca/
</TextBlock>
```

4. In Design view, from the ASP.NET 3.5 Extensions category of the Toolbox, drag the ASP.NET Silverlight control and drop it on `slxaml.aspx`.

5. In the Silverlight's control's Properties window (F4), configure the properties as shown in the following table:

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>235, 235, 235</td>
</tr>
<tr>
<td>BorderColor</td>
<td>Black</td>
</tr>
<tr>
<td>BorderStyle</td>
<td>Dotted</td>
</tr>
<tr>
<td>BorderWidth</td>
<td>5px</td>
</tr>
<tr>
<td>Height</td>
<td>100px</td>
</tr>
<tr>
<td>Source</td>
<td>~/textxaml.xaml</td>
</tr>
<tr>
<td>Width</td>
<td>300px</td>
</tr>
</tbody>
</table>

As you see in Figure 16-3, the Silverlight control generates the required HTML markup and JavaScript at runtime to embed the Silverlight control and point it to the `textxaml.xaml` file.
Playing Windows Media files in Silverlight

Silverlight’s `<Canvas>` element supports a `<MediaElement>` object that you can point to a video file, such as Windows Media (`.wmv`) format. Follow these steps to play a Windows Media file within Silverlight:

1. Add an AJAX Web form named `hostwmv.aspx` to your project.

2. Add a text file named `wmvxaml.xaml` to your project (File ➪ New File ➪ Text File ➪ Add) and start with the following markup:

   ```xml
   <Canvas
     <MediaElement
       Source="http://www.kencox.ca/Xaml_Control.wmv"
       Width="268" Height="404" />
   </Canvas>
   ```

   You can change the `Source` attribute value to point to a Windows Media file on your local system or on the Internet.

   If you’re not using the sample Windows Media file, be sure to change the `Width` and `Height` values to match your video’s size. Otherwise, people may wonder what you’re hiding!

3. In Design view, add the ASP.NET Silverlight control to the page and set its `Height`, `Width`, and `Source` properties to correspond to your Windows Media file’s dimensions and the location of `wmvxaml.xaml`. Here’s some sample markup:

   ```xml
   <asp:Silverlight ID="Silverlight1" runat="server" Height="500px"
     Source="/wmvxaml.xaml" Width="500px" />
   ```

   When you browse to `hostwmv.aspx`, you set off a chain of events: The JavaScript in the page inserts Silverlight that reads `wmvxaml.xaml`, which in turn fetches `Xaml_Control.wmv`. You can see how one control can consume media content from any public site on the Internet.
Silverlight can display other objects while the video plays. For example, the following XAML markup in \textit{wmvxaml.xaml} overlays red text (with white highlight) using <TextBlock> elements:

\begin{verbatim}
<Canvas xmlns="http://schemas.microsoft.com/client/2007">
  <MediaElement Source="http://www.kencox.ca/Xaml_Control.wmv"
                Width="268" Height="404" />
  <TextBlock Canvas.Left="45" Canvas.Top="45" FontWeight="Bold"
              FontFamily="Verdana" Foreground="White" FontSize="20" >
    WMV Sample
  </TextBlock>
  <TextBlock Canvas.Left="47" Canvas.Top="47" FontWeight="Bold"
              FontFamily="Verdana" Foreground="Red" FontSize="20" >
    WMV Sample
  </TextBlock>
</Canvas>
\end{verbatim}

The <MediaElement> object also supports recent versions of Windows Media Audio, streaming audio via Advanced Stream Redirector (.asx) files, and MP3 audio.

\section*{Displaying Rich Media with the MediaPlayer Control}

The ASP.NET Extensions MediaPlayer control (see Figure 16-4) is the fastest way to present Windows Media on an ASP.NET page. I use the word present because the MediaPlayer control frames the content (using Silverlight) into a skinned scene complete with Start, Stop, Pause, Volume, and Full Screen mode buttons.

\begin{figure}[h]
  \centering
  \includegraphics[width=0.5\linewidth]{mediaplayer.png}
  \caption{The ASP.NET MediaPlayer control includes Stop, Start, and Pause buttons.}
  \end{figure}
Unlike the Silverlight control, you don’t need to provide the MediaPlayer control with a .xaml file. Point it to the .wmv location, and you’re done. Follow these steps to display a Windows Media file by using the MediaPlayer control:

1. **Add an AJAX Web form named mediacntrl.aspx to your project.**
   The MediaPlayer control requires the ScriptManager control that’s included in the AJAX Web Form template.

2. **From the Toolbox, in Design view, drop an ASP.NET MediaPlayer control on the page.**

3. **Using the MediaPlayer control’s Properties window (F4), set the MediaSkin property to Professional and the MediaSource property to the location of a .wmv file (for example, http://www.kencox.ca/media_control.wmv).**

At runtime, you can click the MediaPlayer control’s Full Screen button (far left) to expand the display to the screen’s width and height. Press the Escape (Esc) key to return to the browser-embedded view.

### Embedding Flash in an ASP.NET Page

Flash is a hugely popular format for presenting animation and videos on a screen, especially with browser plug-ins. Web sites like YouTube.com use Flash extensively because the download size is reasonable, and almost every browser has the technology or can get it. (You sometimes hear geeks referring to a Flash file as a *swiff*, a name derived from the Flash extension, .swf).

### Downloading and installing Flasher

Technically, you don’t need ASP.NET to display Flash content on a page because the basic requirement is an HTML<object> tag with a bunch of configuration parameters. However, getting all the settings correct is finicky and error-prone. It’s far easier to use an ASP.NET component, such as the excellent Flasher freebie control from my MVP colleague Steve Orr.

Flasher not only handles the configuration, it also adjusts the settings for the best experience based on the browser type. Follow these steps to download and install the Flasher component:

1. **Download the free Flasher control from** [www.steveorr.net/articles/Flasher.aspx](http://www.steveorr.net/articles/Flasher.aspx).

2. **Extract the Flasher.dll assembly from the download into a temporary folder.**
3. Using Windows Explorer, drag Flasher.dll and drop it into the bin folder in Solution Explorer. (Create a bin folder if you don’t have one.)


**Using the Flasher control on a page**

You use the Flasher control like other ASP.NET controls by setting properties. The properties determine the screen size of the Flash movie at runtime. Follow these steps to use the Flasher control on an ASP.NET page:

1. Add a Flash movie to your project (for example, flasherflash.swf from the book’s Web site).

2. From the Toolbox, drop the Flasher control on an ASP.NET page and open the Flasher Properties window (F4), as shown in Figure 16-5.

3. Set the BackColor property to Beige, the Height to 538, and the Width to 739.
   
   You need to adjust the preceding properties according to the Flash movie’s size, but these settings get you started.

4. In the FlashFile property, click the ellipsis button (...) and navigate to the location of the Flash movie (for example, /media/flasherflash.swf or http://www.kencox.ca/media/flasherflash.swf).

   ![Figure 16-5: Setting Flasher's properties.](image)

At runtime, the Flash movie loads and plays. If you use the book’s sample .swf file, you see a motion screen capture created with TechSmith’s Camtasia Studio software.
Ensuring Accurate Rendering with PDF

Adobe’s Portable Document Format (PDF) remains the most popular choice for delivering documents over the Internet. The PDF standard helps ensure the faithful rendering and printing of electronic copies. This section shows a few ways of hosting PDF files in ASP.NET pages. To set up a page for this section’s example, follow these steps:

1. Add a PDF file to your project.
   These examples use a file named clientquotes.pdf.
2. Add an ASP.NET Web form named pdf.aspx to your project.

Rendering PDF within the browser page

The most common scenario is embedding a PDF page within the current browser window. The Adobe Reader plug-in (assuming that it’s installed) replaces the current HTML content and displays the PDF document. Follow these instructions to embed a PDF:

1. From the Toolbox, add an ASP.NET HyperLink control to pdf.aspx.
2. In the Properties window (F4), set the NavigateUrl property to the location of the PDF file — for example, ~/clientquotes.pdf or http://www.kjopc.com/clientquotes.pdf.
3. Set the Text property to Open PDF in this window.

When you click the link at runtime, the generated link opens the PDF file in the browser, leaving the existing browser menu bar and links in place.

Rendering PDF within a new browser page

If you want the PDF to display within the browser but not replace the existing page, follow these steps:

1. From the Toolbox, add an ASP.NET HyperLink control to pdf.aspx.
2. In the Properties window (F4), set the NavigateUrl property to the location of the PDF file — for example, ~/clientquotes.pdf.
3. Set the Text property to Open PDF in new window.
4. Set the Target property to _blank.
You can point the ImageUrl property to an image to create a bitmap link rather than a text link to the PDF.

**Forcing the Open or Save dialog box**

If you want to give the user the option to save a PDF to his computer, you can force the browser to show a dialog box. This technique has the additional advantage of hiding the location of the PDF so that it’s impossible to link directly to it from another Web site. Follow these steps to force the Open or Save option:

1. **Add an ASP.NET Button control named btnOpenOrSave to pdf.aspx.**
2. **In Design view, double-click the button to create the default handler subroutine for the button’s Click event and add the following code in Listing 16-2 to the subroutine.**

```
Try
    Response.Buffer = True
    Response.Clear()
    Response.ClearContent()
    Response.ClearHeaders()  →5
    Response.ContentType = "application/pdf"  →6
    Response.AddHeader("Content-Disposition", "attachment;filename=clientquotes.pdf")  →7
    Response.WriteFile(Server.MapPath("~/clientquotes.pdf"))  →8
    Response.End()  →9
Catch exc As Exception
    Response.Write(exc.Message)
End Try
```

Here’s a walk-through of the code in Step 2:

→**2-5** Instruct the Web server to hold the output in a buffer until it’s all ready. Next, wipe out the HTML markup, headers, and other content that have been generated for this page. (It’s important when forcing a download not to send along anything that the browser might try to render.)

→**6** Set the Response object’s ContentType property to tell the browser to expect a PDF file. The PDF’s Multipurpose Internet Mail Extensions (MIME) type is application/pdf.

→**7-8** Use the Response object’s AddHeader() method to tell the browser to expect an attachment named clientquotes.pdf.
Use the `Server` object’s `MapPath()` method to return the file system path to the PDF file and use the `Response` object’s `WriteFile()` method to push the file out to the browser. On receiving the data, the browser knows that this is a file that can be viewed or saved as a file. It offers the user those options in a dialog box.

You can give the file a different name by changing the text that follows `filename=` in the `AddHeader()` method in Listing 16-2.

### Serving Word on the Web

Older versions of Word acted much like Internet Explorer plug-ins in that they opened files within the browser frame. Newer versions tend to keep to themselves — opening a link to a `.doc` file in a separate Office application.

Consider this use of the ASP.NET `HyperLink` control to point to a Word document:

```xml
<asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="~/clientquotes.doc">Client Comments in Word
</asp:HyperLink>
```

At runtime, when the user clicks the link, Internet Explorer launches the Open or Save dialog box. When the user selects Open, Word fires up and displays the document in read-only mode.

To serve a Word and Excel file programmatically, follow these steps.

1. **Add a Microsoft Word document to your project.**
   
   The file `clientquotes.doc` is used in this example.

2. **Add ASP.NET Web form named `office.aspx` to your project.**

3. **From the Toolbox, add an ASP.NET Button control named `btnServeWord` to the page.**

4. **Double-click the button to create a handler for the Click event and insert this code:**

   ```csharp
   Try
   Response.Buffer = True
   Response.Clear()
   Response.ClearContent()
   Response.ClearHeaders()
   Response.ContentType = "application/msword"
   ```
Depending on where you look, the standard MIME type for Microsoft Word may be given as `application/ms-word` or `application/word`.

You can serve Microsoft Excel files much the same way. You need to change the `ContentType` property to `application/vnd.ms-excel`. Microsoft Office 2007 applications use different MIME types. Check this blog for the variations:


You can deliver many other media types by adjusting the preceding code. Look up the official MIME type (search for MIME type to find a list) and use the type as the `Response.ContentType` value. Don’t forget to change the corresponding filename and extension in the `Content-Disposition` header.
Part IV
Tracking Users, Controlling Access, and Implementing Security

The 5th Wave
By Rich Tennant

“So far our Web presence has been pretty good. We’ve gotten some orders, a few inquiries, and nine guys who want to date our logo.”
In this part...  

ASP.NET ships with a complete set of padlocks for Web pages as well as tools for managing the keys. Chapter 17 builds on an existing sample site to show you how prospective members can register, log in, recover their passwords, and assume a role. You discover how to secure individual ASP.NET pages by changing settings in the configuration file.

Chapter 18 is where I demonstrate how to track the items that users want to buy and then compute the totals and taxes.

Chapter 19 deals with the (potential) evils of user input and how to protect your sites and databases with validation. I show how the built-in validation allows you to examine the data for the right characters, length, and values to render it innocuous. You also discover how the validation controls can alert users to problems without submitting the page to the Web server for processing.
Almost every Web site has something that it wants (or needs) to keep out of the public eye — even if it’s only a page for uploading new content. Because the need to restrict access is so common, the ASP.NET team created a complete set of login, membership, and security management tools.

Understanding Authentication

Authentication is the process of confirming the identity of a person. Say a man arrives at the door to repair your home’s heating system. You need to determine who this person is by looking at some credentials.

The credentials could be a photo identification card plus the fact that the man is wearing company overalls, carrying tools, and arrived in a heating company’s truck. Therefore, you’re satisfied that this person is a heating system repairperson. You’ve authenticated the worker.

However, you haven’t yet authorized him to do any work. You ask others in the house if they’ve called for a furnace repair. To make a long story short, the company’s dispatcher got the street number wrong. The unhappy neighbors are freezing while the technician dawdles at the wrong house!
In the Web site context, users frequently provide their name and password to request authentication. Whatever they offer as credentials must match what’s recorded in your site’s database. After the system establishes their identity, a second mechanism (such as belonging to a specific role) determines whether the person can navigate to some or all of the site’s pages.

Preparing a Site for Membership

ASP.NET’s Membership makes it easy to add logins and security to any site. Instead of building a demonstration Web site from scratch, you build Membership features onto a site that the Microsoft’s ASP.NET team offers as a free download.

Obtaining the Small Business Starter Kit

Microsoft created the Small Business Starter Kit as a sample site for promoting a small business. The license terms allow you to download and alter it to suit your needs. Links change frequently, so here’s the most reliable way to obtain a copy:

1. Navigate to www.asp.net.
2. In the Search box (look for a spyglass), search for the words Starter Kits.
3. From the results, locate the link to download the Small Business Starter Kit and download smallbusiness.vsi to a temporary directory.

If you can’t find the downloadable file, check on this book’s Web site for a link or a copy.

Installing the Small Business Starter Kit

The smallbusiness.vsi file is easy to install by using Windows Explorer. Here are the installation steps:

1. Using Windows Explorer, locate smallbusiness.vsi in the temporary folder where you downloaded it.
2. Double-click smallbusiness.vsi and follow the prompts to install the VB version of the Small Business Starter Kit (see Figure 17-1).

4. When prompted to upgrade the application to use .NET Framework 3.5, click Yes.

Browse to the start page (default.aspx) within Visual Web Developer to confirm that the site is working.

Determining the requirements

Before starting development of the Membership enhancements, it helps to understand the scope of the work by summarizing the requirements:

- Visitors must be able to register for membership.
- Members need a login page and a way to log out.
- Members should be able to recover forgotten passwords.
- The product catalog and the staff area must be available only to registered members. The rest of the site is open to anonymous users.

Now that you know what needs to be done, you can start building the infrastructure and some pages.

Creating the Membership Database

This Web application uses a SQL Server Express database to store user information. The Fabrikam site recognizes three categories of users:

- **Anonymous visitors**: These users don’t need to log in and are not included in the Membership database. They can visit most pages except the catalog and the people pages.
Members: Users who register for access.

Administrators: These are superusers who can add and remove members and generally manage the site. In addition to being registered users, they belong to the Administrators role.

Configuring forms authentication

Forms authentication is the most common type of security for Web sites on the Internet. When users request a restricted page, ASP.NET redirects them to a login page. After the user logs in, the Web server passes the browser a secure, in-memory cookie. Every time the browser requests a secure page from the site, the browser presents the cookie. ASP.NET reads the cookie and, if it’s valid, allows the browser to continue.

The in-memory cookie (one that’s not saved on the user’s hard drive) expires when the user closes the browser. It also expires after 20 minutes (the default) if the user hasn’t requested any more pages from the site. This is why you often have to log back in to a site that you left open during a coffee break.

To configure forms authentication:

1. In Visual Web Developer, choose Website ➪ ASP.NET Configuration.

   The Web Site Administration Tool opens in the browser, as shown in Figure 17-2.

2. Click the Security link.

3. In the lower left of the page, in the Users box (shown in Figure 17-3) click Select Authentication Type.
4. Select the From the Internet radio button and then, in the lower right of the page, click Done.

You may notice later that your project’s web.config file needs to be saved (it’s dirty in geekspeak) even though you haven’t touched it. The Web Site Administration Tool modifies the authentication setting in your site’s web.config file. Here’s what was inserted or changed:

```xml
<authentication mode="Forms" />
```

Many intranet sites use Windows authentication. Internal Web sites and SharePoint applications can accept the same credentials the user provided to log in to his or her computer. The nice part about Windows authentication is that users aren’t bugged to death with dozens of login pages as they go about their work. In geekspeak, this seamless authentication is known as single-sign-on or SSO.

### Creating and enabling a role

*Roles* are labels that identify users as belonging to a group. In this application, you have anonymous users, registered users, and registered users in the Administrator role. Follow these steps to enable and create a role:

1. In Visual Web Developer, open the Web Site Administration Tool (Website ➪ ASP.NET Configuration).
2. Click Security, and in the Roles box (in the lower middle of the Security page), click Enable Roles.
   
   The link changes to Disable Roles. As shown in Figure 17-4, roles are enabled but there are zero roles.
3. Click Create or Manage Roles.
4. In the New Role Name box, type Administrators and click Add Role.
   
   The role appears in the list of roles at the bottom of the page.
Hard to believe, but by adding a role, you create (okay, ASP.NET creates) a new SQL Server Express database in your project. Want proof? In Solution Explorer, refresh the view and expand the App_Data folder. There’s a database called ASPNETDB.MDF and an associated log file.

Implementing Registration and Login

The usual flow of a restricted site is that a user signs up for access by creating a user ID and password. After creating an account, the user can log in and poke around. In this section, you create the sign up (registration), login, and retrieve password pages.

Creating the Registration page with CreateUserWizard

Users create an account by navigating to a sign-up page. This step assumes that you’ve already created the Membership database in the preceding section. Follow these steps to create a registration page for the Small Business Starter Kit site:

1. Making sure to select a master page, add an ASP.NET page named Register.aspx to the starter kit project.

2. From the Login area of the Toolbox, drag a CreateUserWizard control and drop it inside ContentPlaceHolder1, as shown in Figure 17-5.

3. Browse to Register.aspx and fill in the form to create a user.

   You can use the information in the following table if you want reminders of the details in later sections of the chapter.
Field | Value
----- | ----- 
User Name | John Prince 
Password | abc123# 
Confirm Password | abc123# 
E-mail | prince@kencox.ca 
Security Question | What breed is Goldie? 
Security Answer | GoldenDoodle 

4. Click Create User.

The form confirms your registration. If you didn’t create the database, you get an error message. See the previous section, “Creating the Membership Database.”

When complete, the form looks like Figure 17-6.
To align the CreateUserWizard control by using the existing style sheet, I added some markup in bold below. As you can see, the wizard’s default markup is so minimal you’d think that the control doesn’t do much for you. Here’s all there is:

```xml
<div id="content-side1-three-column"></div>
<div id="content-main-three-column">
<asp:CreateUserWizard ID="CreateUserWizard1" runat="server" ActiveStepIndex="1">
    <WizardSteps>
        <asp:CreateUserWizardStep ID="CreateUserWizardStep1" runat="server">
        </asp:CreateUserWizardStep>
        <asp:CompleteWizardStep ID="CompleteWizardStep1" runat="server">
        </asp:CompleteWizardStep>
    </WizardSteps>
</asp:CreateUserWizard>
</div><div id="content-side2-three-column"></div>
```

The CreateUserWizard control is far more sophisticated than it appears from the preceding code because it’s a templated control. This means that you can customize almost everything about its appearance including the prompts and warning messages. (For more on templated controls, see Chapter 13.)

By default, the CreateUserWizard control logs the user in to the site immediately after creating the account. That’s fine for the initial sign-up process, but after that, users need a place where they can log in without creating another account. That’s where you go next.

### Creating the Login page

The Login control is another deceptively simple control with a great deal of power behind the scenes. It sends the user’s credentials to the database and passes an in-memory cookie to the browser if the login is successful. Follow these steps to create a login page.

1. **Making sure to select a master page, add an ASP.NET page named Login.aspx to the starter kit project.**

   The filename is significant because other controls look for it by default.

2. **From the Login category of the Toolbox, drag and drop a Login control inside ContentPlaceHolder1.**

3. **In the Login control’s Properties window (F4), set the following properties to the corresponding values:**
Figure 17-7 shows the effect of setting the preceding properties. There are links to the registration page, the Password Recovery page (you create that next), and a check box to remember the user’s name on subsequent visits.

Figure 17-7: The Login control including optional links.

The Login control handles all the details and error messages. For example, when you run the page and log in with an unknown username, the page displays an error.

Try logging in with the username and password from the previous section “Creating the Registration page with CreateUserWizard”:

User Name: John Prince  
Password: abc123#

A successful login redirects you to the home page, default.aspx. Although you included a link to direct forgetful users to the Password Recovery page, you haven’t created the file yet. That’s your next task.

Creating the Password Recovery page

People forget their passwords so often that armies of Information Technology (IT) professionals are standing by to reset their passwords. It’s a growth industry!
For your site, you use self-serve password recovery where ASP.NET generates a new password and sends it to the user’s e-mail address. To implement the password recovery page, follow these steps:

1. **Making sure to select a master page, add an ASP.NET page named** *Lostpassword.aspx* **to the starter kit project.**

2. **From the Login area of the Toolbox, drag a** *PasswordRecovery control* **and drop in inside** *ContentPlaceHolder1*.

The password recovery process is extremely simple for the user. Type the username, click Submit, and answer the security question. *(GoldenDoodle is the answer if you’ve used the suggested settings.)*

Try recovering a password now, and prepare for a miserable failure when ASP.NET tries to send an e-mail message. Oops! You don’t have a mail server or an e-mail account. The e-mail configuration task is in the next section.

Even though the email message failed, ASP.NET changed the password. The user is locked out. Use the Web Site Administration Tool (Website ➔ Security ➔ Manage Users) to delete and add the user again. Jump ahead to “Adding an administrator” if you need details.

### Configuring the SMTP (Mail) settings

For ASP.NET to send an e-mail message to the user, you require, at the minimum, the address of a Simple Mail Transfer Protocol (SMTP) server that relays e-mail on your behalf. Hosting companies are wary about sending e-mail from an automated process because of abuse by spammers. Therefore, you may have to contact your Web host for specifics as to the SMTP server name, e-mail account name, and password.

After you know the correct values, you can configure the SMTP settings by using the built-in administration tool. Here are the steps:

1. **From Visual Web Developer, choose Website ➔ ASP.NET Configuration.**

2. **Navigate to the Application tab and click the Configure SMTP E-Mail Settings link.**

3. **Fill in the form with the values that your host provides.**

   The following table shows some sample values to help you understand what is required.
Field | Value
--- | ---
Server Name | mail.kencox.ca
Server Port | 25
From | mail@kencox.ca
Authentication | Basic
Sender's user name | mail@kencox.ca
Sender's password | %$whateveritis5&*

### 4. Click Save.

The Configure SMTP Settings page confirms that it saved your settings.

The preceding steps create a `<mailSettings>` element within the `web.config` file's `<system.net>` element. Here's a sample configuration:

```xml
<mailSettings>
  <smtp from="mail@kencox.ca">
    <network host="mail.kencox.ca" password="%$whateveritis5&*"
     userName="mail@kencox.ca" />
  </smtp>
</mailSettings>
```

Password recovery e-mail might fail when you send mail from your development computer, although the same settings work on your public Web site. Some Internet service providers block home users from sending data on Port 25 (the default for e-mail) to deter spam tools.

If you attempt too many incorrect password recoveries using bad SMTP settings, your host may shut down your access pending an investigation. It's better to request the right settings before playing around too much.

After requesting the password, the user receives an e-mail with the subject *Password* and instructions on how to log in.

### Creating a Change Password page

Most of us only change a password when forced to do so. When the breed of your dog works as a password everywhere, why disturb your system? Okay, enough of the bad security practices. To create a page where site members can change their password, add an ASP.NET page named `Changepassword.aspx` to the project. From the Login area of the Toolbox, drag a Change Password control and drop it inside `ContentPlaceHolder1`.
By default, there’s no place to provide a username in the ChangePassword control, so the user must log in first.

If you want to let users change their passwords without logging in, open the Properties window (F4) and change the DisplayUserName property to True.

**Providing a Login/Logout link**

It’s a good practice to end a secure session by logging out rather than just letting the session time out. Additionally, it’s normal to include a link to the Login page. Follow these steps to provide a Login/Logout link on the master page:

1. In Visual Web Developer, open the master page, MasterPage.master in Design view.
2. In the upper right of the page, locate the `<div>` tag with the ID `top-information-home`.
3. From the Login area of the Toolbox, drag a LoginStatus control and drop it just before the Home link, as shown in Figure 17-8.

The LoginStatus control detects whether the user is logged in. If not, it displays the word **Login**, which links to Login.aspx. If the user **is** logged in, it displays Logout and acts as a link button to end the session.

You finally have all the security pages that you need. Trouble is you don’t have any security; nothing prevents an anonymous user from browsing freely throughout the site. In the next section, you lock down some pages.

**Adding an Administration Area**

The administration area is for managing the users and the site. You allow access to the area only to users with the Administrators role. The Small Business Starter Kit doesn’t have built-in administration functions. In this section, you create a simple page as an administration starting point.
Adding the Admin folder and a page

It’s easier to secure entire folders than individual pages. For that reason, all administrative pages go into an Admin folder. Follow these steps to create a folder:

1. In Solution Explorer, create a new folder called Admin (Website ➤ New Folder).
2. Making sure to select a master page, add an ASP.NET page named Memberlist.aspx to the Admin folder.

Next, you create a page for verifying administration security.

Building the Membership List page

As a starter for the Admin section, you allow administrators to view the membership list in a GridView control. Follow these steps to add and configure the GridView:

1. In Design view, from the Data category of the Toolbox, drop a SqlDataSource control into the ContentPlaceHolder area of the Memberlist.aspx page.
2. Using the ASPNETDB.MDF database that you created in the previous section, “Creating the Membership Database,” configure the SqlDataSource according to the following table:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data connection</td>
<td>ASPNETDB.MDF</td>
</tr>
<tr>
<td>Retrieve data from</td>
<td>vw_aspnet_MembershipUsers</td>
</tr>
<tr>
<td>Columns</td>
<td>Email, UserName, LastActivityDate</td>
</tr>
</tbody>
</table>

For help on configuring a SqlDataSource, turn to Chapter 6.

3. Add a GridView control to the ContentPlaceHolder area and set its data source to the SqlDataSource1 from Step 2.
4. Open the GridView control’s Tasks menu and click Edit Columns.
5. In the Selected Fields area, click the LastActivityDate field.
6. As shown in Figure 17-9, in the BoundField Properties area, set the HtmlEncode property to False and the DataFormatString property to {0:MMM dd, yyyy}.

You must turn off HTML encoding to get the DataFormatString to format dates.
7. Click OK.

The Membership database includes several built-in queries called Views. In the preceding steps, you took advantage of a view called `vw_aspnet_MembershipUsers` that returns a list of members.

That’s the only page that you create for the administration function. You’re ready to apply some security so that unauthorized visitors can no longer wander around the site at will.

### Applying Roles and Security

The security requirements for the updated Small Business Starter Kit site allow only registered users to view the catalog pages and the staff personnel pages. Additionally, only members in the Administrators role enter the Admin folder. This section demonstrates two ways to apply security.

#### Securing the Admin folder with roles

The Web Site Administration tool provides a graphical interface for configuring permissions. Follow these steps to allow access to the Admin folder only to members with the Administrators role:

1. Open the Web Site Administration tool (Website ➤ ASP.NET Configuration) and navigate to the Security tab.

2. In the Access Rules box (lower right of the page), click Create Access Rules.
The Add New Access Rule page appears.

3. Using the treeview on the left, select the Admin folder, as shown in Figure 17-10.

4. In the Rule Applies To area, select the Role radio button and from the drop-down list, choose Administrators.

5. In the Permission area, select the Allow radio button and then click OK.

   The view returns to the Security tab.

6. Click Create Access Rules again and select the Admin folder.

7. In the Rule Applies To column, select the All Users radio button.

8. In the Permission column, click Deny and then click OK.

   The Admin folder is now available to members of the Administrators group. Everyone else — denoted by [all] — is barred.

**Understanding access rules**

When you click the Manage Access Rules link, and navigate to the Admin folder, you see a table, such as Figure 17-11, summarizing the rules.

The table shows that the rule for Administrators is on top, and they are allowed. However, the second and third lines have conflicting lines. The [all] group is denied in the middle row and allowed in the grayed-out bottom row.
In case of a conflict, the rule closer to the top wins. That is, the [all] group is denied access. You can use the graphical interface to nudge access rules to create the security settings you need.

The bottom line is dimmed (grayed-out) and can’t be changed here because that rule is inherited from the parent folder, which is the root of the Web.

The management tool stores the rules you just created in a new web.config file within the Admin folder. It looks like the following code, with the <allow> rule for Administrators taking precedence over the <deny> rule:

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.web>
    <authorization>
      <allow roles="Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

### Adding an administrator

In the earlier section, “Creating the Login page,” you registered a regular user, John Prince. To confirm the access rules, you need an administrator. You can add users in the Web Site Administration tool. Follow these steps to create an administrator:

1. Open the Web Site Administration tool (Website ➪ ASP.NET Configuration) and navigate to the Security tab.
2. In the Users box, click Select authentication type.
3. Select the From the Internet radio button and then click Done.
4. In the Users box, click Create User.
5. Fill in the username and other user data.

   Figure 17-12 shows an example account.

6. In the Roles column, select the Administrators check box and then click Create User.

Only members of the Administrators role can browse to pages in the Admin folder. That’s what you need to confirm in the next section.
Confirming the role-based security

If you created the user John Prince in the earlier section, “Creating the Registration page with CreateUserWizard,” you have two accounts to test. Follow these steps to confirm that role-based security is working:

   Instead of viewing the list of members, ASP.NET redirects to Login.aspx.

2. Log in as John Prince (password: abc123#) or another non-administrator.
   ASP.NET redirects the non-administrator to the Login page.

3. Browse to Memberlist.aspx again and log in using an account that belongs to the Administrators group.
   This time, the page appears, including the membership list.

Although the administration area is secure, anonymous users can still browse to the catalog and employee pages. You fix that in the next step.

Securing individual pages

If you had built the Small Business Starter Kit site from scratch, you could have separated the secure and nonsecure pages the same way you did with the administration area by using a folder and a role. Although it’s possible to move the existing files to a new subdirectory, fixing the links requires extra work.
The security plan for the enhanced Web site says that three files in the root directory must be available only to registered, logged in users: Items.aspx, ItemDetail.aspx, and People.aspx. To protect the pages, open the web.config file, locate the closing </configuration> tag, and insert the following XML elements:

```xml
<location path="Items.aspx">
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
<location path="ItemDetail.aspx">
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
<location path="People.aspx">
  <system.web>
    <authorization>
      <deny users="?"/>
    </authorization>
  </system.web>
</location>
```

The <location> element in the web.config file creates an exception for the given path. In this case, the paths are filenames. The embedded elements, <system.web> and <authorization>, specify the areas of the web.config to which the exceptions apply. The final element, <deny users="?"/> , indicates that, for this location, anonymous users (represented by the question mark) are denied access.
Chapter 18

Creating a Shopping Cart with Profiles

In This Chapter
- Using anonymous profiles
- Creating a shopping cart
- Creating classes in VB
- Using inheritance and overloads
- Binding to an ObjectDataSource

People love window-shopping even though it can be a pane when your eyes glaze over. (My apologies for the bad puns. I just wanted to challenge this book’s translators.) I do a lot of pseudo shopping. I wander through the Canadian Tire store in North Bay, picking up interesting tools and gadgets and dropping them into a shopping cart. After pushing these impulse-driven items around in the cart for a while, a nagging inner voice questions whether I have the time or ability to use the tools. More often than not, I sigh and then retrace my steps, putting the tools back on the shelves. I check out with whatever I came for — or sometimes leave without buying anything.

This chapter attempts to bring an enhanced browsing-but-not-shopping experience to a Web site. You’re not required to register or log in but can select items anonymously. Your virtual shopping cart remains intact from one visit to the next. And, unlike the bricks and mortar store, there’s no social pressure to put your non-purchases back where you found them.

Introducing ASP.NET Profiles

Profiles in ASP.NET let you attach data to an individual’s account and retrieve the data on each visit. ASP.NET makes it easy to collect, store, and display
profile information. As this chapter shows, you don’t even need to design a database schema or create a database file. It’s all done within the Visual Web Developer (VWD) environment.

Although this chapter certainly has graphical, drag-and-drop action, you dip further into object-oriented programming with code. You get a feel for using the Profile’s application programming interface (API). Don’t worry; you can follow along quite well without grasping every programming nuance.

Rather than take days to build a realistic Web site from scratch, I show you how to add shopping cart/profile functions to a free sample application from Microsoft. It’s the same site as in the preceding chapter, the Small Business Starter Kit.

There’s no problem if you skipped previous chapters and landed here. There’s no dependency except to refer to some setup instructions. You build a brand-new site based on the Microsoft sample.

**Setting Up the Small Business Sample Site**

The Small Business Starter Kit includes catalog pages based on data in an XML file. To prepare for the shopping cart enhancement, you need to download (it’s free) and install the sample. To get started with the setup, follow these steps:

1. **Download the Small Business Starter Kit from** [www.asp.net](http://www.asp.net).
   
   See Chapter 17 for information on obtaining and installing the Small Business Starter Kit.

2. **Install the Small Business Starter Kit by double-clicking the downloaded file.**

3. **Create a new Web site in Visual Web Developer called** ProfileSite based on the Small Business Starter Kit template (File ➝ New Web Site; choose Small Business Starter Kit under My Templates; and then click Add).

4. **Browse to default.aspx (Ctrl+F5) and other pages in the site to make sure everything is working as expected.**

Although you build on the same base sample you used for the chapter on Membership, this project doesn’t use a login or other Membership features.
Previewing the Final Web Interface

Just as it helps to have a full picture when you’re solving a jigsaw puzzle, it also helps to have a view of the intended result while programming a Web site. In this case, the additions aren’t dramatic; they’re confined to the catalog area and one new page listing the contents of the shopping cart.

The Add to Cart interface

The existing catalog site displays the product’s name, picture, price, description, and availability. Figure 18-1 shows an Add to Cart link. That’s a new link for, er, adding the item to the cart.

![Figure 18-1: The Add to Cart link for a catalog item.](image)

Tracking the cart status

While users add items to the cart, the status appears in the upper-right area of the Items.aspx page. Although not apparent in Figure 18-2, the text is a hyperlink that navigates to the new shopping cart page that you add later in this chapter.

![Figure 18-2: The shopping cart status line links to shopcart.aspx.](image)
Gawking at the cart contents

The only new page in the shopping cart enhancement is shopcart.aspx, and its key component is the GridView control. As shown in Figure 18-3, the grid lists the contents of the cart along with the totals and taxes.

Figure 18-3 also shows that the shopping cart summary grid includes an edit mode in which users can revise the item quantities. The Delete link removes the entire row.

Building the Shopping Cart in Code

Most of the effort in this chapter involves writing code to create and hold shopping cart items. For this, you use object-oriented programming that involves designing your own objects and classes.

The sidebar “A tiny object lesson” offers a quick overview of objects. If the concepts don’t resonate with you right away, don’t worry; it takes time to get your head around these geeky things. Most of us have muddled through all this until at some point the explanations snapped into place. Just follow along and everything will crystallize eventually.

Defining a shopping cart item class

Somehow, you need to re-create in computer code something that resembles an item that you can put into a shopping cart. Fortunately, the people who wrote the Small Business Starter Kit defined an item in code for you. You can examine an item from two perspectives: the data and the schema.
Picking apart an item

The sample Web site stores its catalog data in XML files. Here’s how to get a close-up look at an item:

1. In Solution Explorer, open the App_Data folder.
2. Double-click to open Items.xml.
3. Scroll through the data (past the <category> nodes) to find an <item> element.

   Press Ctrl+G (Go To Line) and enter 193 to find the first <item>.

Here’s the XML that describes the first item:

```
Chapter 18: Creating a Shopping Cart with Profiles

A tiny object lesson

An object is a thingy, a whatchamacallit, a widget, or a thingamabob that sits in the computer’s memory. These whatsis have characteristics (called properties), such as size, color, amount, and ID. The properties carry values, such as Large, Red, 25, and txtTextBox.

Objects can also do stuff. They often contain pre-coded subroutines (methods in geekspeak), such as GrowLarge(), ChangeColor(), AddOneMore(), and Disappear().

Some objects resemble frozen dinners from the supermarket. They’re cooked when you buy them but need some adjustment at mealtime, er, runtime. For example, you change their Temperature property from Cold to Hot by tossing them in the microwave.

In most cases, you don’t use a prefabricated object directly. Instead, you declare a variable that represents a copy of the original object and reserves space in memory. Then you ask the computer to construct the new object for you based on the template version.

Your new object starts life as an exact replica, complete with the original’s properties, default values, and methods. In geekspeak, you create an instance of the object by using the New keyword. (Listen for the phrase newing up when geeks talk about code.) Sometimes you can create an instance and configure its properties in one motion by passing in your customizations as parameters.

An important thing about objects is the way they stand apart from each other. Changes you make to one instance (for example, switching the Visible property to True) have no effect on the original template or other instances of the object in memory.

Programmers assemble their code templates in units called classes. A class is a way of organizing related capabilities and associated code. There’s trouble if two programmers use the same class name and both versions get into the same program; computers freak out over confusion. Fortunately, the compiler detects any mix-up and complains loudly. To avoid naming conflicts and ambiguity, geeks sometimes tack on an additional identifier to their classes called a namespace. (Geeks will tell you that a namespace disambiguates the classes. I’m serious!)

The preceding might sound like gobbledygook to you right now. However, some day, you’ll think back to this chapter and tell yourself, “Oh. That’s what the guy in the book was trying to say!”
```
The <item> element has a formal definition in the project. You can find the formal definition inside Items.xsd located in the App_Data/schemas folder.

If you want to put the preceding item into a shopping cart, you need to know the ID, the title (the item’s name), and its price. You don’t need the description or the item’s image, but you do need to know how many of that item the visitor is ordering. (The item count is missing from the data file because it’s not relevant there.)

To summarize, here’s what you need to track for each item:

- **ID**: An identifier for the item (a string)
- **Title**: The name of the item (a string)
- **Price**: The cost of the item (a number)
- **Count**: How many of this item are being ordered (a number)

**Creating the class file**

You store the class definition in a project, so you need a file. Here’s how to create the class file in a new folder:

1. Right-click the App_Code folder and choose New Folder from the context menu that appears.
2. Name the new folder ShoppingCart (no space).
3. In the ShoppingCart folder, add a class file named CartItem.vb (File ➤ New File ➤ Class ➤ Add).

The class file includes the following starter code:

```vbnet
Imports Microsoft.VisualBasic

Public Class CartItem

End Class
```
The `Imports` statement at the top of the class file helps to incorporate external classes into your program. For example, you can create an instance of the `StringBuilder` class using a long statement like this:

```csharp
Private a As New System.Text.StringBuilder
```

However, if you put `Imports System.Text` at the top of the file, you can shorten the declaration and still get IntelliSense support:

```csharp
Private a As New StringBuilder
```

**Declaring the variables**

Within the body of the class, you need to define class variables (some geeks call them `fields`). In the following code, the keyword `Private` means that these variables are available only to code within this class. Notice that `Itm_Title` (the product name) is a `String` type. `Itm_Price` is declared as a `Decimal` type because it represents fractions of dollars — namely, dollars and cents.

Add the following declarations after the `Public Class CartItem` line:

```csharp
Private Itm_ID As Integer
Private Itm_Title As String
Private Itm_Price As Decimal
Private Itm_Count As Integer
```

**Creating getters and setters**

The `CartItem` class has four properties that you need to declare. Notice how each property corresponds to one of the variables declared in the preceding code.

The `Property` keyword creates a property within a class. `Get...End Get` wraps the code to retrieve the current value of the property. `Set...End Set` assigns a value to the property. The `accessors`, as they’re called, seem needlessly repetitive, but you need them anyway.

If you hear geeks talking about `Getters and Setters`, they’re talking about `get accessors` and `set accessors`.

Add the following code above the `End Class` statement:

```csharp
Public Property ItemID() As Integer
Get
    Return Me.Itm_ID
End Get
Set(ByVal value As Integer)
    Me.Itm_ID = value
End Set
End Property
```
Public Property ItemTitle() As String
  Get
    Return Me.Itm_Title
  End Get
  Set(ByVal value As String)
    Me.Itm_Title = value
  End Set
End Property

Public Property ItemCount() As Integer
  Get
    Return Me.Itm_Count
  End Get
  Set(ByVal value As Integer)
    Me.Itm_Count = value
  End Set
End Property

Public Property ItemPrice() As Decimal
  Get
    Return Me.Itm_Price
  End Get
  Set(ByVal value As Decimal)
    Me.Itm_Price = value
  End Set
End Property

Making a shiny new cart item
You now have a class, class variables, and properties. When you need to create a cart item from the CartItem class, something must construct the cart item for you. That something is a subroutine called New(). Add this marginal little subroutine just before the End Class line:

Public Sub New()
End Sub

When you order a pizza, you give the server some instructions (parameters to geeks) as to what toppings you want. In the same way, the class designer can let you pass along instructions when you order an instance of the class. Add the following code after the previous code:

Public Sub New(ByVal Itm_Id As Integer, ByVal Itm_Title As String, _
                ByVal Itm_Count As Integer, ByVal Itm_Price As Decimal)
  Me.Itm_ID = Itm_Id
  Me.Itm_Title = Itm_Title
  Me.Itm_Price = Itm_Price
  Me.Itm_Count = Itm_Count
End Sub
Ordinarily, the compiler complains if you have two subroutines in the same class with the same name. However, you can get away with it if the two subroutines have different signatures. The signature is the stuff after the word `New` and between the parentheses.

Geeks call a subroutine with a duplicate name but different signature an *overload*. Here in Nipissing Township, we know an *overload* when we see one: It’s Evan’s loaded gravel truck bumping along Alsace Road in the spring before the frost is out.

The preceding overload invites you to send along an identifier, the name (title) of the item, the price, and the quantity as you request a new `CartItem` object.

**Making the class serializable so it travels well**

The `CartItem` class is in good shape but missing the `Serializable()` attribute that makes an object easier to transmit across the network and to store in the Profiles database. Add the attribute and its strange angle bracket syntax as a prefix to the class declaration so it looks like the following:

```
<Serializable()> Public Class CartItem
```

Geeks call what you just added *decorating* the class. If that’s their idea of decoration, don’t invite me to a Big Fat Geek Wedding.

Okay folks, that’s all that’s required to create the `CartItem` class and make it usable in code. Now you need a shopping cart.

**Defining the shopping cart class**

Your computer code shopping cart acts as a container for items just like the metal cart in the hardware store. The computer version has more capabilities that include removing items from the cart, giving you a list of what’s in the cart, and allowing you to change the number of items. In essence, you’re defining a class to manipulate a big list.

**Creating the ShoppingCart class**

The code for the `ShoppingCart` class goes into the App_Code/ShoppingCart folder that you created for the `CartItem` class. Follow these steps to create the file and set up the code:

1. In your Visual Web Developer project, select the App_Code/ShoppingCart folder.
2. Add a class file called `ShoppingCart.vb`.
3. At the top of the file, add the following `Imports` statements:
Imports System.Collections.Generic
Imports System.Linq

By adding the Imports statement, you can use the short name List in your code instead of the long version System.Collections.Generic.List.

4. Before the class declaration, add the Serializable() attribute so the declaration looks like the following:

   <Serializable()> Public Class ShoppingCart

This decorates the class as Serializable() for easy data storage in the Profiles database.

**Inheriting capabilities from a base class**

The ShoppingCart class is all about manipulating the list of items the user has put into the cart. Microsoft has already written tons of code to do this, so you want to tap into work already done rather than write your own.

A common technique is to create your own class but then borrow, or inherit, someone else’s class. It’s like inheriting a furnished house from a rich uncle; you get lots of valuable stuff to use at no cost. You’re not stuck with what you inherit; if the furniture in the bedroom doesn’t suit you, you can substitute your own.

In this project, you inherit System.Collections.Generic.List. If you check the documentation for this class, you find that it can do many interesting things like add, remove, insert, clear, and sort items in a list. The List class is generic in that it takes all kinds of items. You just have to tell it what kind of thing you want the list to hold.

In this case, you want it to hold cart items, which you manufacture in the CartItem class. Therefore, when you inherit the class (known now as the base class), you let it know what to expect. Add the following code in a line below the class declaration (that is, below Public Class ShoppingCart):

   Inherits System.Collections.Generic.List(Of CartItem)

The preceding statement delivers all the capabilities of the List class. You just have to ask for what you want, which you do in the next section.

**Adding items to the list**

Way back in Figure 18-1, you previewed the graphical interface for adding an item to the list. It’s just a LinkButton control with the text Add to Cart. The code that adds an item resides in the AddCartItem() method within the ShoppingCart class. Add the code in Listing 18-1 to the ShoppingCart class (ShoppingCart.vb), just above the closing End Class line.
Listing 18-1: AddCartItem Method in the ShoppingCart Class

```vbnet
Public Sub AddCartItem(ByVal ItemID As String) '1
    Dim pfile As ProfileCommon '2
    Dim cartItem As CartItem '2
    Dim catalogItem As Item '2
    pfile = CType(HttpContext.Current.Profile, ProfileCommon) '5

    If IsNothing(pfile.ShoppingCart) Then '7
        pfile.ShoppingCart = New ShoppingCart '9
    End If '9

    cartItem = _ '11
        pfile.ShoppingCart.FindItemByID(CInt(ItemID)) '11
    If IsNothing(cartItem) Then '12
        cartItem = New CartItem '12
        catalogItem = Catalog.GetItem(ItemID) '17
        cartItem.ItemID = CInt(catalogItem.Id) '17
        cartItem.ItemTitle = catalogItem.Title '17
        cartItem.ItemPrice = catalogItem.Price '17
        cartItem.ItemCount = 1 '18
        pfile.ShoppingCart.Add(cartItem) '19
    Else '21
        cartItem.ItemCount = cartItem.ItemCount + 1 '21
    End If '21

    pfile.Save() '24
End Sub '24
```

Here’s how Listing 18-1 works:

→1 The AddCartItem() subroutine takes one parameter: an ItemID. The LinkButton supplies the ItemID during its Click event so its handler can pass the value along to whoever wants the information. Notice that ItemID is a string, such as 07. That becomes significant later in the code.

→2 This line probably has you asking, “What the heck’s a ProfileCommon?” This is a class that ASP.NET creates for you at runtime so you can use all the wonderful features of Profiles.

The page won’t compile at this stage, and you’ll see VWD griping about ProfileCommon and any code that refers to it. That’s because you haven’t configured the <profile> section of the web.config file yet to tell ASP.NET about your needs. If the error messages are really bugging you, jump ahead to the section “Enabling profile data and anonymity in web.config” to fix the web.config file.

→5 To use a ProfileCommon object, you need to declare a variable, pfile, to hold it. You don’t want just any ProfileCommon object; you want the one for the current browser. (Otherwise,
you’d store one person’s preferences in another person’s record.) This line gets it for you.

The `HttpContext.Current.Profile` represents the current user’s profile, but it needs to look like a `ProfileCommon` object. The solution is to cast the object into the correct type using the `Ctype()` method.

→7-9 You check whether the user has a shopping cart in his/her profile. If not (`pfile.ShoppingCart is Nothing`), go build a new ShoppingCart and put it in the profile.

→11 The next sequence adds an item to the shopping cart. However, it’s possible that the item is already in the cart. This line uses a helper function called `FindItemByID()` (you create it later in this chapter) to snoop inside the shopping cart to see whether the item’s there.

→12-18 The `If` statement assumes that no item with the same `ItemID` value is in the cart, so you have to add one. First, create a basic `cartItem` object using the `CartItem` class you built earlier in this chapter. Wait a minute! The only thing you know about this item is that it has an ID of some sort. You need details such as the price. Fortunately, a routine in the Small Business Starter Kit called `GetItem()` fetches a catalog item if you tell it the `ItemID`.

When you call `Catalog.GetItem()`, you get back a `catalogItem` object from which you can extract the `Id`, `Title`, and `Price` properties. You can turn those into properties of the `cartItem` object. The quantity of items doesn’t come from the catalog, so you set the `ItemCount` property to 1.

→19 Now that you’ve configured the shopping item, you need to put it into the list of items. Recall that you inherited from the generic `List` class. `List` has an `Add()` method that’s just sitting there waiting to be used. You don’t need to write your own `Add()` method; just provide the `cartItem` object that you want to include.

→21 This line represents some unfinished business. It handles the case where the `ItemID` already exists in the profile. Rather than create a new `cartItem` object, you just want to bump the total count by one.

→24 Having created (or modified) the `cartItem` object and inserted it into the list, you need to store the data. The `ProfileCommon` object handles the task for you with its `Save()` method.

### Removing items from the list

Deleting an item from the list is far easier than adding an item because you don’t need to fetch any details about the item. Add Listing 18-2 to `ShoppingCart.vb`. 
Listing 18-2: DeleteCartItem Method in the ShoppingCart Class

```vbnet
Public Sub DeleteCartItem(ByVal ItemID As Integer)
    Dim pfile As ProfileCommon
    pfile = CType(HttpContext.Current.Profile, ProfileCommon)
    Dim cartItem As CartItem
    cartItem = pfile.ShoppingCart.FindItemByID(CInt(ItemID))
    If Not IsNothing(cartItem) Then
        pfile.ShoppingCart.Remove(cartItem)
        pfile.Save()
    End If
End Sub
```

You saw most of Listing 18-2 in the preceding section, “Adding items to the list.” The code looks into the shopping cart for a cart item with the ItemID value that the calling routine passed in. If the search turns up an object fitting the description, use the generic List class’s built-in Remove() method to delete the item from the list. The ProfileCommon object’s Save() method pushes the changes into the database.

### Finding an item in the shopping cart

In Listing 18-1 and Listing 18-2, I skipped past the helper function FindItemByID(). This code (shown in Listing 18-3) uses a LINQ query to return the CartItem object with the matching ItemID. The keyword Me refers to the current instance of the class. In this case, Me gets the currently executing instance of the ShoppingCart so the query can snoop inside for CartItem objects. If the routine finds a matching ItemID, it returns the CartItem object.

Add Listing 18-3 to the ShoppingCart class above the End Class line.

Listing 18-3: Using LINQ to Find an Item Inside a Shopping Cart

```vbnet
Public Function FindItemByID(ByVal ItemID As Integer) As CartItem
    Dim q = From ci In Me Where ci.ItemID = ItemID
    Return q.FirstOrDefault
End Function
```

### Getting the list of items

When you construct the GridView control that displays the items in the shopping cart, you need something to fill the grid. The GetCartItems() function fills the bill by getting a reference to the current user’s profile and, within it, his or her ShoppingCart object. Add Listing 18-4 to the class in ShoppingCart.vb.
Listing 18-4: Getting All Cart Items

```vbnet
Public Function GetCartItems() As List(Of CartItem)
    Dim pfile As ProfileCommon
    pfile = CType(HttpContext.Current.Profile, ProfileCommon)
    Return (pfile.ShoppingCart)
End Function
```

**Updating the number of items**

As you saw in Figure 18-5, the user can update the quantity of an item using the GridView control by going into edit mode and typing a number. In Listing 18-5, the subroutine takes two parameters: the ID of the item to update and the number of items. These values are passed in from the GridView row that’s being edited.

Although the GridView provides a Delete link, there’s another way of removing an item: Enter 0 or a negative number as the count. The routine checks for a value less than 1 and calls the DeleteCartItem routine from Listing 18-2.

Listing 18-5: Updating the Item Count

```vbnet
Public Sub UpdateItemCount(ByVal ItemID As Integer, ByVal itemCount As Integer)
    If itemCount < 1 Then
        DeleteCartItem(ItemID)
        Exit Sub
    End If
    Dim pfile As ProfileCommon
    Dim itm As CartItem
    pfile = CType(HttpContext.Current.Profile, ProfileCommon)
    itm = pfile.ShoppingCart.FindItemByID(ItemID)
    If Not IsNothing(itm) Then
        itm.ItemCount = itemCount
        pfile.Save()
    End If
End Sub
```

Most of the code in Listing 18-5 repeats lines that you’ve already analyzed. The goal is to use the ItemCount property for the item and set it to the number of items that are passed in to the routine. The ProfileCommon object’s Save() method moves the updated value into the database.

Don’t forget to add Listing 18-5 to the class in ShoppingCart.vb.

That’s all that’s required to build the ShoppingCart class. In the next section, you tell the Profile object about these objects.
Enabling profile data and anonymity in web.config

ASP.NET generates the ProfileCommon class for you based on its knowledge of the data you intend to store. Where does it find out about that data? In the web.config file.

Add Listing 18-6 to the web.config file just before the closing </system.web> tag. Adding this markup enables the profile features, defines the data to store in a user’s profile record, and turns on anonymous identification. It also stops VWD from complaining about ProfileCommon in the classes you created previously.

Listing 18-6: Enabling Profile Data and Anonymous Identification

```xml
<profile enabled="true" defaultProvider="AspNetSqlProfileProvider">
    <properties>
        <add name="ShoppingCart" type="ShoppingCart"
             serializeAs="Xml" allowAnonymous="true"/>
    </properties>
</profile>
<anonymousIdentification enabled="true" cookieless="UseCookies"
                          cookieName=".ASPXANONYMOUS" cookieTimeout="100000"
                          cookiePath="/" cookieRequireSSL="false"
                          cookieSlidingExpiration = "true" cookieProtection="None"
                          />
```

Here’s a breakdown of the profile attributes that Listing 18-6 inserts in each user’s profile:

- **name**: Lets you identify the object by name in code as in `pfile`. `ShoppingCart`.
- **type**: Refers to the name of the class that holds the data.
- **serializeAs**: Instructs ASP.NET to format the object as XML for storage. Recall that you marked the `ShoppingCart` class as `Serializable` so this could readily convert to XML.
- **allowAnonymous**: Makes this content available to users who haven’t registered for the site. Just like in the Canadian Tire store, you can wander and pick up items without anyone asking your name!

When you set the preceding `allowAnonymous` attribute to `true`, you must also enable anonymous identification. Therefore, you include the `anonymousIdentification` element and set its `enabled` attribute to `true`.
Updating a Web Page to Add Profile Data

In the preceding section, you created the plumbing that adds, removes, and updates a shopping cart. The ASP.NET Profile object takes care of storing each user’s shopping cart in the database.

The great thing about adding a shopping cart to Microsoft’s code (that is, the Small Business Starter Kit) is that Microsoft did the design work. In this section, you find a spot to insert a LinkButton control that adds an item to the user’s shopping cart.

Inserting a LinkButton into the page

It can be tricky to update a page without messing up an existing layout. Follow these instructions to add the LinkButton markup with a minimum of collateral damage:

1. In the Small Business Starter Kit project, open Items.aspx.
2. In Design view, near the bottom of the page, locate the GridView control with the ID of GridViewItems. (Be careful because there’s also a GridViewCategories that you don’t want to disturb.)
3. From the GridView control’s Tasks menu, choose Edit Templates.
   The ItemTemplate template appears, as shown in Figure 18-4.

4. Insert the cursor to the right of the Read More hyperlink and type a space, a pipe character (|), and another space. Leave the cursor where it is.
5. In the Standard category of the Toolbox, double-click a LinkButton control to insert the control at the cursor location.
Configuring the LinkButton control

The Items.aspx page already binds to a data source. This means you can wire up the new LinkButton as a child control. Follow these steps to configure the LinkButton control:

1. **In the LinkButton control’s Properties window (F4), set the ID value to lnkbtnAddToCart and the Text property to Add to Cart.**

2. **In the LinkButton control’s Properties window, click the lightning bolt icon to bring the events into view.**

3. **In the Command event, enter AddToCart as the handler name.**

4. **Open the LinkButton control’s Smart Tasks menu and choose Edit Databindings.**

The DataBindings dialog box appears, as shown in Figure 18-5.

5. **In the Bindable Properties area, select the CommandArgument property.**

6. **As shown in Figure 18-5, in the Binding for Command Argument area, select the Custom Binding radio button and type the following expression:**

   ```
   Eval("ID")
   ```

7. **In the Bindable Properties area, select Enabled and enter the following custom binding expression:**

   ```
   IIf(Eval("InStock")= true, "true", "false")
   ```

8. **Click OK.**

The preceding steps use two inline statements:

- The first step tells the LinkButton to get its CommandArgument data by binding to the page’s data source and pulling the value from the ID field.
The second inline statement combines two statements. The inner function, `Eval("InStock")`, gets the value of the InStock field from the data source. The `IIF()` function tests whether the value of InStock is true. If true, `IIF()` emits the string true as the value of the Enabled property. If InStock turns out to be false, the string false effectively disables the control. (There's no point allowing users to add an item to the shopping cart if you don’t have any in stock.)

If you look at the LinkButton in Source view, the markup resembles Listing 18-7.

Listing 18-7: The AddToCart LinkButton

```xml
<asp:LinkButton
    ID="lnkbtnAddToCart" runat="server" CommandArgument='<%# Eval("Id") %>'
    Enabled='<%# IIf(Eval("InStock")= true, "true", "false") %>'
    Text="Add to Cart" oncommand="AddToCart">
</asp:LinkButton>
```

**Adding the LinkButton event handler**

You told the LinkButton control to run the AddToCart routine when the Command event fires. It’s time to supply the handler code. Follow these steps to insert the handler:

1. **Open** Items.aspx.vb.
2. **Just above the** End Class statement, **insert the code in Listing 18-8.**

The event handler in Listing 18-8 creates an instance of the ShoppingCart class that you built in the preceding section, “Defining the shopping cart class.” With the object, it calls the AddCartItem method, sending along a command argument. The command argument is the item’s ID as recorded by the LinkButton.

Listing 18-8: The oncommand Event Handler

```vbnet
Protected Sub AddToCart(ByVal sender As Object, ByVal e As CommandEventArgs)
    Dim scart As New ShoppingCart
    scart.AddCartItem(e.CommandArgument.ToString())
    Response.Redirect("~/shopcart.aspx", True)
End Sub
```

After launching the routine to add the item, the built-in `Response.Redirect` method redirects the browser to a page you create in the next section, shopcart.aspx.
Building a Page to Manage Cart Contents

To complete the shopping cart enhancement of the Small Business Starter Kit, you need a page where users can view and change the contents of the shopping cart. In this section, you create the data connection and add a DataGrid.

Adding the shopcart.aspx page

The Small Business Starter Kit uses a master page to present the common page elements and styles. Follow these steps to add a new page that follows the existing site’s layout and column structure:

1. Making sure to select the site’s master page, add a new page (File ➪ New File ➪ Web Form) named shopcart.aspx to the project.
2. In Source view, just before the closing \</asp:Content> tag, add the markup shown in Listing 18-9.

Listing 18-9: Replicating the Column Structure

```html
<div id="pagetitle" style="right: 0px; top: 0px">
  Shopping Cart
</div>
<br />

The markup in Listing 18-9 is purely cosmetic. It generates the three-column layout and indicates the location of the GridView that you add later in this chapter.

Adding an ObjectDataSource to handle data

The ObjectDataSource control is like the other ASP.NET data source controls in that it fetches, adds, and deletes data on behalf of other controls.
However, the ObjectDataSource doesn’t work directly with a database. Instead, it talks to methods in the ShoppingCart class that you created in the preceding section, “Defining the shopping cart class.” Follow these steps to add and configure an ObjectDataSource to the shopping cart page:

1. With shopcart.aspx open in Design view, from the Data category of the Toolbox, drag an ObjectDataSource control and drop it in the ContentPlaceHolder area.

2. In the Properties window (F4), change the ObjectDataSource ID to objDSourceShoppingCart.

3. From the Tasks menu of the ObjectDataSource control, choose Configure Data Source.

4. In the Choose a Business Object screen, from the drop-down list, choose ShoppingCart and then click Next.

5. In the Define Data Methods screen, on the Select tab, from the drop-down list, choose the following option:

   ```csharp
   GetCartItems(), returns List<CartItem>
   ```

6. On the Update tab, choose the following option:

   ```csharp
   UpdateItemCount(Int32 ItemID, Int32 itemCount)
   ```

7. On the Delete tab, choose the following option:

   ```csharp
   DeleteCartItem(Int32 ItemID)
   ```

8. Click Finish.

When called on to perform a data operation, the ObjectDataSource hands off the task to one of the routines in the ShoppingCart class. In Source view, the markup looks like Listing 18-10. (If you’re wondering about inserting data, that’s done in the preceding “Configuring the LinkButton control” section.)

**Listing 18-10: Markup for the ObjectDataSource**

```xml
<asp:ObjectDataSource ID="objDSourceShoppingCart" runat="server"
    TypeName="ShoppingCart" SelectMethod="GetCartItems"
    UpdateMethod="UpdateItemCount" DeleteMethod="DeleteCartItem">
    <DeleteParameters>
        <asp:Parameter Type="Int32" Name="ItemID" />  
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Type="Int32" Name="ItemID" />
        <asp:Parameter Type="Int32" Name="ItemCount" />
    </UpdateParameters>
</asp:ObjectDataSource>
```
Adding a GridView and using the ObjectDataSource

The GridView control is handy for letting users view and update data. To save space, the following steps show how to create a functional but minimalist version of the grid:

1. From the Data category of the Toolbox, add an ASP.NET GridView control to the ContentPlaceHolder area of shopcart.aspx.
2. In Source view, use the markup in Listing 18-11 to create a minimalist version of GridView1.

Listing 18-11: Minimalist GridView

```xml
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False" DataKeyNames="ItemID" DataSourceID="objDSourceShoppingCart">
  <Columns>
    <asp:CommandField ShowDeleteButton="True" ShowEditButton="True">
    </asp:CommandField>
    <asp:BoundField ReadOnly="True" DataField="ItemID" Visible="False"/>
    <asp:BoundField ReadOnly="True" DataField="ItemTitle" HeaderText="Title"/>
    <asp:BoundField DataField="ItemCount" HeaderText="Qty."/>
    <asp:BoundField DataField="ItemPrice" HeaderText="Price" DataFormatString="{0:C}"/>
  </Columns>
</asp:GridView>
```

The shopping cart isn’t complete without showing the costs including taxes. That’s covered in the following section.

Creating a Calculations class

The average real-life shopping cart (you know, the ones with at least one sticky wheel that makes you look like you can’t steer) doesn’t update you on the cost of your items. Web-based carts can track the item totals, subtotals, and taxes. Follow these steps to implement a Calculations class to generate the figures:

1. In the App_Code/ShoppingCart folder of your project, right-click and add a new class called Calculations.vb.
2. Replace the existing code with the code in Listing 18-12.
As you see in the following snippet, the `CalcSubTotalPrice()` function uses a LINQ `Aggregate` clause to look at the shopping cart items and the `Sum` function to compute the total dollar value:

```vbnet
subtotalPrice = Aggregate ci In pfile.ShoppingCart.GetCartItems() _
    Into Sum(ci.ItemPrice * ci.ItemCount)
```

To calculate the taxes, the `CalcOntTax()` and `CalcGST()` functions request the subtotal and multiply that by the percentage tax. (Ontario’s provincial tax is 8%, and Canada’s Goods and Services Tax is 6%. Be sure to replace my outrageous tax rates with yours. The values are marked in bold in Listing 18-12.)

The `CalcTotalPrice()` function sums the various costs and returns them as the `totalprice`, a decimal number.

**Listing 18-12: Calculations.vb**

```vbnet
Imports Microsoft.VisualBasic
Imports System.Web.Profile.ProfileBase
Imports System.Linq
Imports ShoppingCart

Public Class Calculations
    Public Shared Function CalcSubTotalPrice() As Decimal
        Dim subtotalPrice As Decimal
        Dim pfile As ProfileCommon
        pfile = CType(HttpContext.Current.Profile, ProfileCommon)
        subtotalPrice = Aggregate ci In pfile.ShoppingCart.GetCartItems() _
            Into Sum(ci.ItemPrice * ci.ItemCount)
        Return subtotalPrice
    End Function

    Public Shared Function CalcSAndH() As Decimal
        Return 5.5
    End Function

    Public Shared Function CalcOntTax() As Decimal
        Return (CalcSubTotalPrice() * 0.08)
    End Function

    Public Shared Function CalcGST() As Decimal
        Return (CalcSubTotalPrice() * 0.06)
    End Function

    Public Shared Function CalcTotalPrice() As Decimal
        Dim totalprice As Decimal
        totalprice = CalcSubTotalPrice()
        totalprice = totalprice + CalcSAndH()
        totalprice = totalprice + CalcOntTax()
        totalprice = totalprice + CalcGST()
        Return totalprice
    End Function
End Class
```
Inserting Calculations columns

The final step is to put the subtotal, taxes, and total cost on the Web page. You start with the GridView code from Listing 18-11 and add two template columns that extend into the footer. Follow these steps to insert two columns:

1. Open the GridView control created in the preceding section, “Adding a GridView and Using the ObjectDataSource.”
2. In the Properties window (F4), set the ShowFooter property to True.
3. In the Columns property, click the ellipse button to open the Fields dialog box.
4. In the Selected Fields area, select the Price field, set its Visible property to False, and click OK.
5. Switch to Source view, and just before the GridView control’s closing tag, insert the markup in Listing 18-13.

That completes the development of the shopping cart. The final section of the chapter walks you through the features you’ve built.

Listing 18-13: Adding Totals in the Footer

```xml
<asp:TemplateField>
  <ItemTemplate>
    <%=Format(Eval("ItemPrice"), "C").PadRight(256).Substring(0, 256)%>
  </ItemTemplate>
  <FooterTemplate>
    Subtotal<br />
    S&amp;H<br />
    Ont. Tax<br />
    GST<br />
    TOTAL
  </FooterTemplate>
  <HeaderTemplate>Price</HeaderTemplate>
</asp:TemplateField>

<asp:TemplateField>
  <ItemTemplate>
    <%=Format(Eval("ItemCount") * Eval("ItemPrice"), "C").PadRight(256) _
      Substring(0, 256)%>
  </ItemTemplate>
  <FooterTemplate>
    <%=Calculations.CalcSubTotalPrice(), "C"%><br />
    <%=Calculations.CalcSAndH(), "C"%><br />
    <%=Calculations.CalcOntTax(), "C"%><br />
    <%=Calculations.CalcGST(), "C"%><br />
    TOTAL
  </FooterTemplate>
  <HeaderTemplate>Total</HeaderTemplate>
</asp:TemplateField>
```
Walking Through the Shopping Cart Profile

It’s time to tour your enhancements to the Small Business Starter Kit. The Web application is great for anonymous visitors because they don’t need to register to add items to their cart.

Adding items to the cart

Follow these steps to add an item to the shopping cart using the Web interface:

1. **Browse to** default.aspx, and **choose Items from the main menu**.
   The list of categories appears.

2. **Click the hyperlink for the Amet category**.
   The subcategories of the Amet category appear.

3. **Click the second subcategory, Vestibulum**.
   The items in the subcategory appear.

4. **In the Pellentesque item listing, click the Add to Cart link**.
   The Shopping Cart page appears with the item in the GridView.

You now have an item in the shopping cart. You can repeat the sequence to add more items or continue to the next section to update the quantity.

Updating the quantity of an item

The Add to Cart link adds only one item per click to the shopping cart. However, you can edit the number on the summary page. Follow these steps to test the update capability:

1. **Browse to** shopcart.aspx.

2. **In the GridView, click the Edit link on the row you want to update**.

3. **In the ItemCount column, type the quantity of the item and then click the Update link**.

After each update, the calculation revises the subtotal, taxes, and total cost.
In This Chapter

- Why user input is evil
- Checking for a range of values
- Avoiding cross-site scripting hacks
- Being a regular expression kind of person
- Escaping with your HTML

Is there anyone who *likes* filling in forms? Paper forms are the worst. In the passport office, you can line up for hours while desperately hoping that you’ve entered the right information. At the business counter, you hold your breath as a clerk checks the fields in your passport application. While she ponders too long over an answer, you fear that you’re living the old game of Snakes and Ladders (also known as Chutes and Ladders). Providing an unsuitable answer is like landing on a ‘Snake’ square: Fate sends you sliding helplessly back to the wrong end of a growing queue.

Web forms aren’t that much easier than the passport office. You dutifully and diligently fill in a dozen text boxes on a page. You click Submit, and . . . #$%^&! . . . your answers disappear into a blank screen! Trying not to panic, you click the browser’s Back button to recover your input, only to read that the page has expired. Expired? It was fresh only a second ago!

Meanwhile, on the other side of the server, Web developers don’t like forms either. Users don’t pay attention to what the form is asking. People enter bad data, no data, and sometimes, malicious data. Validating their responses takes a lot of effort and you’d rather leave it until the end.

The ASP.NET team provides controls that take some of the pain out of validation for users and programmers. This chapter shows you how and where to use validation. Like with most chapters in this book, the goal is to write as little code as possible.
**Remembering User Input Is Evil**

Accepting data in a public Web form is a risky business with many potential pitfalls. The bad guys out on the WWW (Wild West Web) use data input fields as prime methods (*attack vectors* in geekspeak) to disrupt Web servers and run their own nasty code. After they worm their way in, they steal information, deface sites, and use the computer as a zombie to send spam and even more malicious content.

The best defense is not to allow *any* data into your Web application until you check the data and cleanse it. If you start with the assumption that anything a user can submit to your site is *evil until proven otherwise*, you’re more likely to implement appropriate validation to stay out of trouble.

Your advantage over the hackers is that for every text box on a page, you know exactly what data is acceptable in terms of business rules and data integrity. You know whether you’ll accept letters from A to Z and how many. If you’re requesting a number, you know the valid range, such as 1 to 150. The validation controls help you enforce your requirements.

**Forcing the User to Enter Something**

Most forms have required fields that the user must not leave blank. For example, a login page can’t get far without some sort of identifier, such as a username. The `RequiredFieldValidator` control insists that the user enter *something* in the field, even if it’s gibberish.

The validator is hard to fool; spaces don’t count as an entry. What’s more, the control recognizes default text and treats that particular something as nothing. Follow these steps to add and configure the `RequiredFieldValidator` control:

1. **From the Toolbox, add a TextBox control to an ASP.NET page and set its Text property to Enter Your User ID.**
   
   *Enter Your User ID* is an instruction to the user and can’t be counted as a username.

2. **Add a Button control to the page.**

3. **Add a RequiredFieldValidator control to the page and set the following properties to their corresponding values:**
Open the page in the browser and click the Submit button. Even though text is in the text box, the InitialValue property of the RequiredFieldValidator signals that the user must enter something other than the default text.

Setting the Display to Dynamic avoids a postback by using JavaScript to display the error message. The SetFocusOnError property enhances usability by putting the cursor into the TextBox control where the error occurred.

Here's the markup that's generated by the preceding steps:

```html
<asp:RequiredFieldValidator
  ID="RequiredFieldValidator1" runat="server"
  ControlToValidate="TextBox1"
  InitialValue="Enter Your User ID"
  SetFocusOnError="True"
  Display="Dynamic"
  ToolTip="Enter your user name"
  Text="* Type something else!">
</asp:RequiredFieldValidator>
```

If you want to include text in a text box to prompt the user, check out the AJAX Control Toolkit’s TextBoxWatermark control. Its prompt text disappears when the user starts typing. For more information on enhancing pages with the AJAX Control Toolkit, see Chapter 15.

### Ensuring That a Value Is within a Range

The RangeValidator control ensures that the user enters a value that falls within the set minimum and maximum. For example, in an online ordering page, you wouldn’t allow negative values, such as –10 or ridiculous quantities such as 2 million. Follow these steps to check for a range of numbers:
1. Add a TextBox control and a Button control to an ASP.NET page.

2. From the Validation category of the Toolbox, drop a RangeValidator control on the page.


4. Set the ControlToValidate property to the text box you added (for example, TextBox1).

5. Set the Type property to Integer.

   This ensures that the control validates the input as a number, not a string.

If you enter a number outside the range of 1 to 10000, the validator complains. Likewise, if you enter Four, the validation fails.

Here’s the markup generated by the preceding steps.

```html
<asp:RangeValidator ID="RangeValidator1" runat="server" ControlToValidate="TextBox1" Display="Dynamic" MaximumValue="10000" MinimumValue="1" ErrorMessage="Must be 0 to 10000" SetFocusOnError="True">
</asp:RangeValidator>
```

Although the RangeValidator handles ranges of dates, be careful about the date format. The value 11/12/2008 means November 12 or December 11, depending on the culture. For best results, have the user select a date from a calendar control. An excellent AJAX calendar can be found in the ASP.NET Control Toolkit (see Chapter 15).

The RangeValidator has an It's not my job mentality and only bothers to check what the user enters. If the user doesn’t provide a value but leaves the text box blank, the RangeValidator allows the validation to pass. Therefore, you probably need the RequiredFieldValidator control in addition to RangeValidator. Check out the previous section, “Forcing the User to Enter Something,” to add a RequiredFieldValidator control.

Setting the Type property correctly is critical to the success of the RangeValidator — especially when you check numbers. If you leave the Type set to String in the preceding example, the user can slip 100,000 right past the validator’s nose. Trouble is, String is the default, so it’s easy to forget.
Checking and Comparing Values

The CompareValidator control offers three validators in one: compare values in two ASP.NET controls, compare a value in a control against a constant value, and test whether a user has entered a valid data type.

The available operators are Equal, NotEqual, GreaterThan, GreaterThanOrEqual, LessThan, LessThanOrEqual, and DataTypeCheck.

The operator names are self-explanatory. For example, GreaterThanOrEqual validates when the input value is greater than or equal to a second control’s value (or a constant value).

Comparing values in two controls

The CompareValidator can determine whether the value in one text box is greater than the value in a second text box. Follow these steps to display an error message if the validation test fails:

1. Add two ASP.NET TextBox controls to a Web form.
2. Add a Button control to the page.
3. From the Validation category of the Toolbox, add a CompareValidator to the page.
4. In the CompareValidator control’s Properties window, set the following properties and corresponding values:
As shown in Figure 19-1, the CompareValidator complains at runtime if the value in the first text box isn’t greater than the value in the bottom text box.

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ControlToCompare</td>
<td>TextBox2 (or whatever your second text box is named)</td>
</tr>
<tr>
<td>ControlToValidate</td>
<td>TextBox1 (or whatever yours is called)</td>
</tr>
<tr>
<td>Display</td>
<td>Dynamic</td>
</tr>
<tr>
<td>ErrorMessage</td>
<td>The top ^ must be greater!</td>
</tr>
<tr>
<td>Operator</td>
<td>GreaterThan</td>
</tr>
<tr>
<td>SetFocusOnError</td>
<td>True</td>
</tr>
<tr>
<td>Type</td>
<td>Double</td>
</tr>
</tbody>
</table>

Making the CompareValidator dynamic

Sometimes you need to validate a control’s value against a value from a database or other source. For example, you may want to exclude lawyers who charge more than a $2,000 per hour from applying for work, but the exclusion level can vary. Follow these steps to set the constant and the error message:

1. Add an ASP.NET TextBox control to a Web form.
2. Add a Button control to the page.
3. From the Validation category of the Toolbox, add a CompareValidator to the page.
4. In the CompareValidator control’s Properties window, set the following properties and corresponding values:
5. In Design view, double-click an empty area of the page to create a handler for the Page Load event.

The event handler code appears in Source view.

6. Within the Page_Load subroutine, add the following code:

```vbnet
Const decMaxFee As Decimal = 2000
Dim strErrMsg = FormatCurrency(decMaxFee, 2)
CompareValidator1.ValueToCompare = decMaxFee
CompareValidator1.ErrorMessage = strErrMsg & " max!"
```

The code sets the maximum fee to 2000 and formats the value as currency (dollars in the en-us culture) with two decimal places. The second to last line assigns the maximum fee as the comparison constant. The last line builds an error message by using the formatted amount and a bit of extra text. (In geekspeak, adding strings of text together is concatenation.)

## Checking a data type

The CompareValidator control’s DataTypeCheck operator can help you determine whether a user entered a valid number or date. The validation can be quite picky about what it considers a date. For example, December 24, 2008 is a valid date value (especially when it’s your birthday!); however, the CompareValidator rejects the date format as insufficiently geeky. Follow these steps to test for a valid date:

1. Add an ASP.NET TextBox control and a Button control to the page to a Web form.

2. From the Validation category of the Toolbox, add a CompareValidator to the page.

3. Set the ControlToValidate property to the ID of the text box.

4. Set the Operator property to DataTypeCheck and the Type to Date.
As shown in Figure 19-2, the validation routine is smart enough to know that 13/13/2008 is bogus because no 13th month exists. Although the CompareValidator catches bad dates, it’s better to maintain tight control over date input. For example, show the user a calendar picker or use a masked input control. See Chapter 15 for other sophisticated input controls.

![Figure 19-2: Testing for an invalid date.](image)

**Diversion:**

DataTypeCheck can validate for a string type, but it’s hard to find something on a keyboard that won’t pass as a string. Numbers, dates, and even spaces are valid strings to the CompareValidator.

**Using the RegularExpressionValidator**

A word of warning about this section: regular expressions are weird and extremely geeky. Here’s one that might scare you away:

```
\w+([-+.']\w+)*@\w+([-+.\w]*\w+([-+.\w]+)*
```

Microsoft supplies the preceding example with the RegularExpressionValidator control to verify (and enforce) the format of an e-mail address. Some of us find the rules of the Klingon language (of Star Trek fame) easier to grasp than the syntax in regular expressions.

The idea of regular expressions is that you create groups of rules that match or don’t match characters at the beginning, middle, and end of a string. Sometimes, a regular expression limits the number of characters; sometimes, it requires a certain number of characters; and other times, it doesn’t matter.

**Testing for one, two, or three numbers**

Many books and Web sites are dedicated to regular expressions and many tools can help you assemble them. For your purposes, assume that you figured out — or someone sent you — the following regular expression that checks the validity of an Item ID. The Item ID starts with a digit and ends with a digit. It can have one, two, or three digits but no other characters. Here’s the regular expression:
Follow these steps to implement the preceding regular expression in a RegularExpressionValidator control:

1. Add an ASP.NET TextBox control and Button control to the page.
2. From the Validation category of the Toolbox, add a RegularExpressionValidator control to the page.
3. Set the ControlToValidate property to the ID of the text box.
4. Set the ErrorMessage property to 1-3 digits please.
5. Set the ValidationExpression to: \^[0-9]{1,3}\$

When you browse to the page, try combinations of characters that aren’t digits, such as 1a1, or try too many digits, as shown in Figure 19-3. The only way to avoid entering the required match is to leave the text box empty. That signals you may also need the RequiredFieldValidator to handle this input (see the earlier section, “Forcing the User to Enter Something” to do so).

Checking the length of text in a multiline text box

When you set the ASP.NET TextBox control’s TextMode property to MultiLine, you can’t limit the length of text that a user enters. It ignores the MaxLength property because the control generates an HTML Textarea control that doesn’t have a Length property.

The danger is that a user could submit huge amounts of text to overwhelm the system and cause an error that sophisticated code could exploit. You can avoid this problem with a regular expression that validates the length of the text. Follow these steps to check the length of a multiline text box:
1. Add an ASP.NET TextBox control to a Web form.

2. Set the TextMode property to MultiLine.

3. From the Validation section of the Toolbox, drop a RegularExpressionValidator control on the page.

4. Set the ControlToValidate property to TextBox1 (or whatever you named your control) and the ErrorMessage value to Maximum 250 Characters.

5. Set the ValidationExpression property to the following value:

   ^\[sS\]{0,250}$

At runtime, the user can still enter huge amounts of text. However, Figure 19-4 shows that the regular expression won’t let the user submit the text if it’s longer than 250 characters. Adjust the allowed length by changing 250 to whatever length is acceptable.

---

**Validating Data with Code**

Validation can be complex in a business application where the rules can change frequently. You may need to read data from several controls on the page, query a database, or contact a Web service to determine if the user’s input is valid. For example, after a customer enters a postal code, you can check whether the closest warehouse has sufficient quantity to ship the product by the chosen date.

The CustomValidator calls client-side and server-side functions that you provide — no built-in freebies in this validator! In this example, you test the users input and simulate a server-side routine that checks the available quantity. Instead of creating a separate client-side validation, I show you how the ASP.NET AJAX UpdatePanel control validates without a full page refresh.

Follow these steps to implement a CustomValidator control:

1. From the AJAX Extensions category of the Toolbox, add a Script Manager control and an UpdatePanel control to the ASP.NET page.
2. Inside the UpdatePanel control, drop a TextBox control and a Button control onto the form.

3. Drop a CustomValidator control on the page.

4. In the CustomValidator control's Properties window, set the ControlToValidate property to TextBox1 (or whatever you named your control) and the Display value to Dynamic.

5. In Design view, double-click the Button control to create a handler routine for its Click event and add the following code as the handler:

   ```vbnet
   Protected Sub Button1_Click ByVal sender As Object, ByVal e As System.EventArgs
   If Page.IsValid Then
       Response.Redirect("thanks.aspx")
   End If
   End Sub
   ```

6. In Design view, double-click the CustomValidator control to create a handler routine for the ServerValidate event and use the following code as the handler:

   ```vbnet
   Protected Sub CustomValidator1_ServerValidate ByVal source As Object, ByVal args As System.Web.UI.WebControls.ServerValidateEventArgs
   Dim intQty As Integer
   intQty = HowManyWidgets()
   If Convert.ToInt32(args.Value) > intQty Then
       args.IsValid = False
       CustomValidator1.ErrorMessage = "Only " & intQty.ToString & " available."
   Else
       args.IsValid = True
   End If
   End Sub
   ```

7. In Source view, add the following function to simulate a call to the database:

   ```vbnet
   Public Function HowManyWidgets() As Integer
   Return 13
   End Function
   ```

Browse to the page, enter 30 in the text box and click Submit. The validation event fires on the server. As shown in Figure 19-5, the error message reports the exact problem — not enough widgets.

The Page.IsValid property in Step 5 checks the status of the validation controls (you have only one) on the page. If all the controls report that their input is okay, the IsValid property is True and the routine continues. In this case, the page redirects to a (nonexistent) Thank You page.
Step 6 inserts code that executes whenever CustomValidator1 verifies its associated text box. The routine determines how many widgets are available in the factory by calling the HowManyWidgets() function. Unlike in a real app, the function doesn’t actually check anything and always reports 13 widgets. (You can’t get good help nowadays!)

Here, the args parameter is a ServerValidateEventArgs object. Inside that object is a Value property that you want to test. After converting the value to a number, the code checks to see if the user has requested more items than what are available. If the order is too large, the routine sets the IsValid property to False and generates an error message. If everything is fine, it sets IsValid to True so that everything can proceed.

The CustomValidator control can test an empty field. To evaluate missing input, you need to set the ValidateEmptyText property to True.

Validating by Groups

A busy Web form — especially one that uses ASP.NET AJAX — can have numerous text boxes and buttons that are independent of each other. Instead of trying to validate every control on the page at the same time, you can validate controls in groups. To include controls in a group, set their ValidationGroup property to a common value.

Figure 19-6 shows the Design view of two independent validation groups on the same ASP.NET page. The upper section checks for a number from 1 to 10. The lower section includes a required field. At runtime, click OK to validate the number even though the required field is empty.

All the controls in the upper section include the ValidationGroup= "Range" attribute and value pair. The controls in the lower section are grouped by the common "Required" value.
Displaying a Summary of Invalid Fields

The ValidationSummary control collects error messages from the validation controls on the page and displays those errors as a bullet list, a regular list, or a paragraph of regular text — your choice. Figure 19-7 shows how to provide a longer explanation of the error in the validation summary and a shorter error message adjacent to the control. The longer message comes from the control’s ErrorMessage property. The short message is assigned to the validation control’s Text property.

If you’re validating by group name with the ValidationGroup property, you must specify the group name in the ValidationSummary control. Otherwise, error messages from the group won’t appear.
If you want to punish users for failing to enter valid data, set the ValidationSummary control’s ShowMessageBox property to True. This shoves a JavaScript Alert box in the person’s face. An Alert box can’t be ignored. On many PCs, it makes a beep to wake the person and teach him to pay more attention. To continue using the browser, the user must click OK to dismiss the Alert message (see Figure 19-8). (In geekspeak, its window is application modal.)

Seriously, Alert boxes are annoying, insulting, user-hostile, and unnecessary in most cases — including validation. It’s better to reserve the shock treatment for actions that have serious, irreversible results, such as formatting the hard drive or wiping all the data from a database.

![Figure 19-8: A punishing Alert box.](image)

**Defanging Markup for Safety**

If you want to accept HTML markup in a text box, you must disable ASP.NET’s built-in defense against JavaScript attacks. (See the previous sidebar, “ASP.NET tries to protect you”). To disable the protection you change the Page directive to look like this:

```html
<%@ Page Language="VB" ValidateRequest="false"%>
```

You can ward off much of the danger of script attacks by encoding the HTML before it gets into your database. The Server object’s HTMLEncode() method converts troublesome characters into their escaped format. In geekspeak, they’re now entities.

Here’s a little demonstration that might convince you.

1. Add an ASP.NET page named defang.aspx to your project.
2. Add a TextBox control and a Button control to the page.
3. Double-click the Button control to create a default handler for its Click event and insert the following line of code in the subroutine:

```csharp
Response.Write(TextBox1.Text)
```
4. Disable the protection against scripting attacks by changing the `Page` directive to look like the following:

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

5. Browse to the page, type the following into the text box, and click the button:

```<script>location.href='http://kencox.ca';</script>```

You see that if the malicious script got into your database and displayed on a page, visitors could be redirected to a site of the attacker’s choice.

6. Change the code used in Step 3 to the following:

```Response.Write(Server.HtmlEncode(TextBox1.Text))```

7. Repeat Step 5.

The malicious script has been defanged and looks like this in the browser’s source code:

```&lt;script&gt;location.href='http://kencox.ca';&lt;/script&gt;```

All user input is evil until proven otherwise.
Part V
Getting the Bugs Out and Handling Runtime Errors

The 5th Wave
By Rich Tennant

“Okay, I think I forgot to mention this, but we now have a Web management function that automatically alerts us when there’s a broken link on The Aquarium’s Web site.”
In this part... 

If you write twice as much code as the previous day, do you create twice the number of bugs or do you just double the chances of introducing a single bug? Fortunately, I don’t deal with theoretical possibilities in this part. Instead, Chapter 20 shows you practical ways to find out what’s going wrong by using the Visual Web Developer debugging tools. In Chapter 21, I discuss how to make your perfect pages more robust so they don’t collapse into a sorry pile of electrons when something beyond your control interferes with their normal operation.
Chapter 20

Debugging and Tracing Pages

In This Chapter

- Design-time errors and squiggly lines
- Compiling errors with the compiler
- Debugging logic errors
- Setting breakpoints and values
- Tracing problems to the root

The following scene happens to all of us:

Someone (perhaps a boss) finally expresses an interest in what you’ve been working on for the last week or so and asks for a quick demonstration. The visitor sits at your keyboard (drives in geekspeak) and starts exploring as you look on, beaming. Suddenly, disaster strikes! Your beautiful page displays an ugly error message:

Object reference not set to an instance of an object.

The visitor politely offers to come back another time while you turn to find out what happened. Welcome to the world of bugs and debugging. This chapter explores the features of the integrated development environment’s (IDE) debugger and the tools that give you an inside look at your code while it executes.

Note: The Express version of VWD doesn’t include some of the debugging features described here. I’ve included a note each time so you’ll know.

Setting Up an Error Page Scenario

To follow along with the explanations of the Debug toolbar and numerous debugging windows, it helps to have a common sample of problematic code. Most of this chapter uses Listing 20-1, so type or paste the code into an ASP.NET file. Some of the problems in Listing 20-1 are immediately obvious and others are more subtle.
Listing 20-1 doesn’t work as listed. The code is supposed to parse a few lines of text, extract words that have more than two characters, and insert the selected words into a drop-down list. When the user clicks OK, the selected word should appear in the Label control at the top of the page (as shown in Figure 20-1).

Listing 20-1: Problem Code to Debug

```vbcn
<%@ Page Language="VB" %>
<script runat="server">
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    If Page.IsPostBack = True Then
      Dim words() As String
      words = getText()
      Dim capWords = From word In words Select word 
        Where word.Length > 2 Order By word
      For Each word In capWords
        ddl.Items.Add(word.ToLower)
      Next
    End If
  End Sub

  Protected Sub btnOK_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    lblText.Text = ddl.SelectedValue
  End Sub

  Function getText() As String()
    Dim sb As StringBuilder
    Dim splts() As Char = {",", ".", "!", " ", Chr(34)}
    sb.Append("You mumble something like, ")
    sb.Append(""It's never done that for me. ")
    sb.Append("It must be a bug!"")
    sb.Append("Your triumphant mood is gone.")
    Return sb.ToString.Split(splts, StringSplitOptions.RemoveEmptyEntries)
  End Function
</script>
</html>
```
Analyzing Design-Time Errors

Visual Web Developer (VWD) watches while you code (or paste code) to alert you to errors that could cause trouble at runtime. The background compilation looks for syntax errors and flags them in Source view with a squiggly underline. Look closely at the first `sb.Append` function in Figure 20-2; VWD has flagged it as a problem.

Pass your mouse over the squiggle to see a tooltip containing the IDE’s complaint. Here’s the tooltip text:

```
Variable 'sb' is used before it has been assigned a value. A null reference exception could result at runtime.
```

For more detail on any error, including the file and exact line and column number, open the Error List pane (View ➤ Error List), as shown in Figure 20-3.
For now, resist the temptation to get rid of the aforementioned squiggle. You need the error in the next few pages to highlight features of the debugger.

Double-click the error description to jump to the line that’s causing the error.

Some errors that appear in Source view are even more prominent when you switch to Design view. For example, consider the following code snippet (it’s not part of Listing 20-1):

```xml
<xsp:DropDownList
    ID="ddl" runat="server" CssClass="largecntrl">
</xsp:DropDownList>
```

In Design view, the designer won’t deal with the code at all. The error message appears in a big gray box. You need to fix problems like this manually in Source view.

### Discovering Compile-Time Errors

Complex Web sites often use third-party controls, such as advanced grids for specialty purposes. Add-in controls might depend on assemblies that aren’t yet in your project. Visual Web Developer might not notice every dependency at design-time, but you can’t fool the compiler. While building, the compiler insists on locating every called assembly in the code.

### Building a single page

The compiler catches errors on a page-by-page basis. To invoke the compiler, right-click a page in Solution Explorer and from the context menu, choose Build Page. The validation results appear in the Output pane, as shown in Figure 20-4.
Click the Toggle Word Wrap button on the Output pane (shown in Figure 20-4) to show the build results without the need to scroll horizontally.

**Building a whole Web site with exclusions**

You can build a whole Web project to test for errors. Right-click the project root and choose Build Web Site.

Visual Web Developer can exclude buggy or incomplete pages from the build process. In Solution Explorer, right-click a page you want to exclude and choose Exclude From Project from the context menu. VWD adds an .exclude extension to the filename so that it won’t appear as an ASP.NET file.

To “unexclude” a file, right-click and choose Include In Project.

**Finding Logic Errors**

Visual Web Developer has loads of sophisticated tools for monitoring, dissecting, and troubleshooting your code. The hard part might be knowing which tool is best for the task. In this section, you set breakpoints, step through code, examine the errors list, and use the various debugger windows.

**Analyzing the sample page at runtime**

The best way to check if a page runs is to, er, run it. Browse to the page that you created from Listing 20-1 — right-click the file in Solution Explorer and choose View in Browser. The result is depressingly underwhelming. The drop-down list appears but nothing’s in it. This confirms a bug exists. The debugger helps you find it.

**Setting a breakpoint in the code**

A breakpoint pauses the execution of a program so that you can analyze the logic. When you execute a page in the debugger, Visual Web Developer compiles the code, launches the browser, and runs the code until it reaches a breakpoint.

In your buggy example, code within the handler for the Page Load event is supposed to fill the drop-down list. The Load event is a good place to look for an error. Follow these steps to set a breakpoint:
1. Open the sample ASP.NET page in Source view.

2. Locate the handler code for the Page Load event (it’s near the top of the source code).

3. Click in the gray margin to the left of the code.

   A red dot appears and the IDE highlights the corresponding line of code, as shown in Figure 20-5. That red dot marks the breakpoint.


   If you’re debugging the project for the first time, a dialog box appears with an offer to enable debugging. Click OK.

   The IDE builds the page, launches the browser, and then returns to the source code at the location of the breakpoint.

5. Choose Debug: Step Into (or press F11).

   The arrow advances to the next line of code, which is an If statement, as shown in Figure 20-6.

---

**Test-driven development**

Large-scale, enterprise Web projects often use test-driven development where a developer writes testing code to prove that each subroutine works as expected. The accumulated tests stay with the code for repeated runs to catch bugs that slip in during further development. Geeks call errors that appear in previously working code regression errors or regressions.

Test-driven development is a valuable technique, although you may feel that writing tests is overkill on small projects where code quality isn’t as critical as it would be in an online banking application.
6. Watch the code carefully and advance the execution another step (F11).

Yikes! The debugger jumped straight to the End If statement, bypassing the code that fills the drop-down list.

7. End the debugging session by choosing Debug ➤ Stop Debugging.

You know the location of the problem (which is half the battle), but not why the code failed.

**Examining values while debugging**

While you step through code in the debugger, Visual Web Developer generates a huge amount of information about what’s occurring. Follow these steps to examine the values of objects and to discover why the code didn’t execute:

1. **On the sample ASP.NET page, run the debugger (F5) to the breakpoint.**
   
   You set the breakpoint in the preceding section, “Setting a breakpoint in the code”.

2. **Step into the next statement (F11) that has the following line of code:**
   
   ```vbnet
   If Page.IsPostBack = True Then
   Dim words() As String
   words = getWords()
   Dim sizeText =
   ```

3. **Hover the mouse pointer over the IsPostBack property.**
   
   As shown in Figure 20-7, the value of Page.IsPostBack pops up and it is False.

4. **Close the debugger (Debug ➤ Stop Debugging).**

   The problem is clear: Page.IsPostBack is False when a page runs for the first time (there hasn’t been a chance for a postback to happen). However, the
code's *If* statement is looking for *IsPostBack* to be *True*. It's a simple logic error where the test is reversed.

The programmer's intention was to fill the drop-down list only when the page loads for the first time. (Filling a drop-down list on every page load creates duplicate and triplicate items.) The fix is to change the *If* line to the following:

```vbnet
If Page.IsPostBack = False Then
```

---

**Tracking Down a Runtime Error**

This section analyzes an error that brings down a running page. When a page crashes at runtime, the compiler tells you what it knows about the problem. The information is usually helpful — but beware that it doesn't always point to the real source of the error. It often points to where it *discovered* the error.

When you run the test page again you get farther but, as shown in Figure 20-8, not far enough. The page *seems* to have crashed in the `getText()` function because of a missing object instance.
To track down the error with the debugger, follow these steps:

1. **Open the ASP.NET page in Source view.**
2. **Delete any existing breakpoints (Debug ▸ Delete All Breakpoints; click OK).**
3. **Set a new breakpoint (F9) at the following line:**
   ```vbscript
   Function getText() As String()
   ```
   For more on breakpoints, see the previous section, “Setting a breakpoint in the code.”
4. **Start debugging (F5) and wait until the execution stops at the breakpoint.**
5. **Open the Locals pane (Debug ▸ Windows ▸ Locals).**
6. **Watch the values in the Locals pane and step into the function (F11).**
   - As shown in Figure 20-9, the variable `sb` (supposedly a `StringBuilder` object) has a value of `Nothing`. The code is about to use the `Append()` method, so `sb` must be *something*. `Nothing`, in this case, is a sign of trouble.

![Figure 20-9: The sb object is Nothing but must be something.](image)

7. **Take one more step (F11) in the debugger.**
   - Sure enough, the code crashes. This time the error message offers more help, as shown in Figure 20-10. The `New` keyword is missing.
8. **Stop debugging and consider the new information.**

   The problem is that the programmer declared the `sb` variable a `StringBuilder` but forgot to create an instance of the object (*instantiate* in geekspeak) using the `New` keyword — a common mistake. It’s like declaring your *intention* to build a flight of stairs from your house’s main floor to the basement. Despite your best *intentions* you haven’t actually installed the stairs. Oops! Someone just tried to walk down the nonexistent stairs and crashed to the basement floor!
It's hard to tell when you need to use `New`. Sometimes you use an object directly without a variable as in `Response.Write()`. Some methods return a fully “New” object for you without using `New`. Worse, LINQ syntax makes everything weird because it infers objects for you from the context without declaring them.

The fix is to create the object so that it looks like this line:

```vbs
Dim sb As New StringBuilder
```

At a conference of geeks, you often hear the presenter say, “I’ll go ahead and New up that variable,” while she types a line of code. Presenters love to say, “Go ahead.”

With the fix, the page functions properly. The drop-down list fills with words longer than two characters. When you click OK, the selected word appears in the label. Mission accomplished.

## Breaking Based on a Condition

The technique of setting a breakpoint and stepping through the code can be time consuming, especially in a big `For...Each` loop involving tons of data. An alternative is to watch for a given value or condition. When that value appears, break into the debugger for a look.

**Note:** Visual Web Developer Express does not support this feature.

In the following steps, I show you how to use the working (bug fix) version of Listing 20-1 to break into the debugger when a certain value appears.

Follow these steps to break based on a condition:

1. Open the ASP.NET page in `Source view`. 
2. Add a breakpoint on the following line:

```csharp
ddl.Items.Add(word.ToLower)
```
For more on breakpoints, see the previous section, “Setting a breakpoint in the code.”

3. Right-click the breakpoint (the red circle) and choose Condition from the context menu that appears.

4. Enter the following condition statement inside the Breakpoint Condition dialog box.

```csharp
word.ToLower="triumphant"
```

5. Click OK.

6. Start the debugger (F5) and wait for the code to hit the breakpoint.

7. Hover the mouse over `word` to confirm that the current value is `triumphant`.

8. Hover the mouse over `ddl` until the plus sign (+) appears, and then expand the plus sign to reveal the list of properties.

A list of the object’s current properties and values appears.

---

**Editing a Value during Execution**

A good debugger (a person, not a tool) is part detective. When code misbehaves, it sometimes helps to reenact the crime at the scene by using different values. Perform *what if* scenarios by breaking into the debugger, changing values, allowing execution to continue, and observing the result.

*Note:* Visual Web Developer Express does not support this feature.

Follow these steps to edit a value during the execution of an ASP.NET page:

1. In Visual Web Developer, open a debugged version of Listing 20-1.
2. In the `btnOK_Click` event handler, place a breakpoint at the `End Sub` line.
3. Run the page in the debugger (F5) and click OK.
   The program breaks into the debugger at the `End Sub`.
4. Open the Autos window (Debug ➤ Windows ➤ Autos).
5. Double-click the `lblText.Text` line, shown in Figure 20-11, and change the value inside the quotation marks to the following:

```csharp
&#169 2008
```
6. Continue running the code (F5).

The label displays the changed markup rather than the text of the item chosen in the drop-down list. See Figure 20-12.

You find that the ability to experiment with values in the debugger saves time compared to stopping the entire process, rewriting the code, recompiling, and waiting for the browser to display the result.

Figure 20-12: The result of changing a value while debugging.

**Panes to Ease the Pain**

Visual Web Developer has several panes to help you monitor what’s occurring with your code. Many programmers don’t know how to use them because they’re visible only while you run the debugger or use break mode.

- **Autos** (Debug ➪ Windows ➪ Autos): Displays variables in the current statement and the three statements before and after the current statement. *(Note: The Autos pane is not available in the Express version of VWD.)*

- **Locals** (Debug ➪ Windows ➪ Locals): Acts similar to the Autos pane except that Locals shows the variables that are within the current context.

- **Watch** (Debug ➪ Windows ➪ Watch 1, 2, 3, 4): Monitors up to four variables and expressions. For example, if you type the variable sb into Watch 1, you see the variable sb transition from a Nothing StringBuilder to a full object. Tracking an object with Watch resembles a hot real estate market — the values of the properties are constantly changing.
Immediate (Debug ➪ Windows ➪ Immediate): Handy for testing or changing a value. In the following command, the question mark (?) character means print what follows:

```csharp
? DateTime.Now.AddYears(4)
```

Tracing the (Mis)Steps of a Web Page

`Tracing` is collecting information about an ASP.NET page or application while it runs. You trace data to investigate problems and slow performance in a production site. ASP.NET raises numerous events while it runs. When trace is on, details about those events appear in the trace log.

In this section, you add your trace information to the trace log.

Implementing trace in a page

You trace a single page or all the pages in a Web site.

To trace a single page, open the ASP.NET page in Source view and change the Page directive to include two additional attributes:

```
<%@ Page Language="VB"
    TraceMode="SortByCategory"
    Trace="true" %>
```

Run the page. ASP.NET inserts rows and columns of information about what happened, when it happened, and what the values were. The control tree category is especially useful. The grid describes the controls on the page and shows where they fit inside their parent containers. For example, Figure 20-13 shows `lblText`, `ddl`, and `btnOK` inside `form1` (where they belong).

Check the ViewState Size column if an ASP.NET page seems sluggish and bloated. In many cases, you can put a control on a diet by setting its `EnableViewState` property to `false`.

Figure 20-13: Part of the control tree.
Don’t leave tracing on in a production environment. Apart from it looking weird on the page, the detailed information is valuable to hackers. If you require trace logs on a live site, consider using the Application Trace Overview page.

**Implementing trace for a whole site**

You can enable trace for all the pages on a site by adding the line to the `web.config` file just after the `<system.web>` element:

```xml
<trace enabled="true" pageOutput="true"/>
```

Rather than add trace information to each page, view the data via a special virtual page. As an option, restrict the use of the application tracing to those who are browsing locally. Follow these steps to view application-wide tracing information:

1. In the `web.config` file, add the following element after the `<system.web>` element:
   ```xml
   <trace enabled="true" pageOutput="false" localOnly="true"/>
   ```

2. Browse to an ASP.NET page that you want to trace and leave the browser open.

3. On the browser’s address bar, replace the name of the page with the following `trace.axd` handler name:
   ```http://localhost:3740/debugging/trace.axd```

4. On the Application Trace Overview page (shown in Figure 20-14), click the View Details link to view a file’s trace data.

The Application Trace page stores the last 10 page requests. Change the number by setting the `requestLimit` attribute to the number of pages you want, as shown in the following bold text:

```xml
<trace enabled="true" pageOutput="false" localOnly="true" requestLimit="20"/>
```
Using the Debugger Keys and Toolbar

If you’re new to VWD, you might start debugging by using the Debug menu items. We all have our own styles, but many of us slip into using keystrokes for efficiency. By using the function keys in Table 20-1, you won’t take your eyes off the code and debugger panes during a debugging session.

If you prefer clicking buttons to debug (see Figure 20-15), detach the Debug toolbar and place it near the code where it’s handy.

<table>
<thead>
<tr>
<th>Keystroke</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F5</td>
<td>Start/Continue Debugging</td>
<td>Runs (or continues running) the current page in the debugger. If you’ve used the Set As Start Page command on a different file, it runs that one.</td>
</tr>
<tr>
<td>Shift+F5</td>
<td>Stop Debugging</td>
<td>Ends the debugging session and closes the browser instance.</td>
</tr>
<tr>
<td></td>
<td>Break All</td>
<td>Pauses the execution and enters break mode. Try using this if your code’s caught in a loop.</td>
</tr>
<tr>
<td>Ctl+Shift+F5</td>
<td>Restart</td>
<td>Restarts execution of the page without forcing a recompile.</td>
</tr>
<tr>
<td>F11</td>
<td>Step Into</td>
<td>Execute statements one at a time, including called functions and methods.</td>
</tr>
<tr>
<td>F10</td>
<td>Step Over</td>
<td>Execute statements one at a time except execute called methods and functions as an uninterrupted block.</td>
</tr>
<tr>
<td>Shift+F11</td>
<td>Step Out</td>
<td>Finish executing the remaining statements in the current function or method without stopping.</td>
</tr>
<tr>
<td></td>
<td>Show Next Statement</td>
<td>Move the insertion point to the line of code to execute next. It’s not always obvious that anything has happened when you use this.</td>
</tr>
</tbody>
</table>
Figure 20-15: The Debug toolbar buttons.
Chapter 21

Avoiding Crashes by Handling Exceptions

In This Chapter
- Using the language of exceptions
- Creating a better error page
- Sending error alerts by e-mail
- Trying and catching exceptions
- Fixing some common exceptions

If anything can go wrong, it will. Despite your best efforts to keep bugs out of your code and account for all eventualities, Murphy’s Law applies fully to computer programs. A hidden flaw never stays hidden for long. Code that was working fine for months suddenly acts up. Everyone involved swears nothing changed. Obviously, something changed — it’s just that nobody knows what it was. Sometimes you never find out because the problem mysteriously resolves itself. You eventually dismiss it as a “computer thing” and forget it.

In Chapter 20, you use the VWD debugger and tracing features to root out annoying bugs. In this chapter, you try to cope with the unforeseen and limit the damage when the unforeseen inevitably happens. Remember: Nature always sides with the hidden flaw!

Understanding Exceptions and Their Messages

An exception is the result of an unexpected or unusual situation within a program. Many exceptions are the result of mistakes in programming logic. Others stem from a failure to account for all possibilities, such as when you try to divide a number by zero. Exceptions also occur beyond the programmer’s control, long after he deploys the code to a production site. For
example, a server administrator inadvertently removes a file or changes a setting in Internet Information Services (IIS), and an otherwise stable Web site starts misbehaving.

You read geekspeak throughout this book and this chapter about exceptions is no, um, exception. Although technodudes generally have an aversion to sports, exception terminology tends to be sports related:

- **Throw**: Cause an exception. This aggressive term implies a degree of vigor because it insults and criticizes the quality of another geek’s code, as in, “It not only sucks, it throws.” This term’s keyword is **Throw**.

- **Raise**: A more polite way of expressing the same thing as *throw*. When a Canadian geek doesn’t want to be confrontational about a colleague’s error, she says, “I’m sure you know this already, but your subroutine raises an exception that causes the unfortunate and untimely termination of my program, eh.”

- **Catch**: The act of retrieving details about an error to cope with an undesirable situation. For example, a geek might say, “The whole site went down because of an uncaught error. I hope they catch the guy who wrote that code.” This language is popular enough to warrant its own keyword, **Catch**.

- **Rethrow**: After catching an exception, a geek might decide that some other part of the code is better suited to handle it. One technique is to pass the exception details to the **Throw** keyword, which repeats the exception.

- **Handle**: Doing something about an exception before it comes to the attention of the user or the boss. The words *unhandled exception* strike fear in the hearts of programmers because they failed to cope with the unforeseen and now someone knows they’re not perfect.

- **Eat**: Making an exception disappear and acting like it never happened. From the movie scenes where bad guys swallow incriminating evidence just when the police arrive. A geek might not want the user to see that an exception occurred but isn’t otherwise concerned about it. Some consider eating an exception poor-programming practice because it hides a problem that could make the application unstable.

- **Trap**: Connotes that the error handling (see *catch*) is worthy of praise because of the extra effort involved. Think of baseball where a catcher saves the day by trapping a wild pitch.

- **Wrap**: Implementing exception handling where something could go wrong. For example, a geek might advise, “Wrap that in a Try...Catch block to stay out of trouble.”
Global Error Handling

In ASP.NET, an unhandled exception isn’t pretty. You see it often when you develop and test pages. The details of the exceptions change, but the red Server Error (shown in Figure 21-1) always grates. In this section, you find options for handling errors on an ASP.NET Web site.

Although error stack traces (the exception details) in pages are helpful for developers, it’s not polite to foist them on users (refer to Figure 21-1). Additionally, telling hackers too much about your system and paths is dangerous because the bad guys might use the information to worm their way in. In fact, hackers try to cause errors just to exploit them. Ensure that users don’t see any details by providing a user-friendly, generic error page. Follow these steps to configure a custom error page.

1. Add an ASP.NET page that deliberately creates an error condition.
   Use the following code to throw the exception shown earlier in Figure 21-1.

   ```csharp
   Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
       Dim q = From FileName In System.IO.Directory.GetFiles ("c:\doesntexist\", "*.", System.IO.SearchOption.AllDirectories)
   End Sub
   ```

2. Add `<%@ Import Namespace="System.Linq" %>` to the top of the .aspx page to keep the LINQ query happy.

3. Add an HTML page called oops.htm to your project (File ➤ New File ➤ HTML Page).
Using an HTML page instead of an ASP.NET page is safer because when something wrong occurs with the Web server’s ASP.NET processing, regular HTML might still be working.

4. Add some polite text to the page, as shown in Figure 21-2.

![Figure 21-2: Polite text on a generic error page.]

5. Open the site’s `web.config` file and add the following just before the closing `</system.web>` tag:

   ```xml
   <customErrors mode="On" defaultRedirect="oops.htm"/>
   ```

   When you run the broken page, ASP.NET detects the error and redirects the browser to `oops.htm`. This doesn’t handle the exception — or even tell you that it’s happening — but at least you’re not inflicting your error’s gory details on visitors.

   Two other values for the `mode` attribute come in handy. If you’re trying to track a problem on a production site and need to see the error message, set `mode="RemoteOnly"` and browse to the page on the local machine via `http://localhost/`. You see the error locally, but remote users (that is, Internet users) still go to the redirect page. The other setting, `mode="Off"`, disables custom error handling.

   Custom error pages give visitors information that corresponds to what happened. For example, if someone browses to a nonexistent page, you can detect the Web server’s 404 status code and redirect him to a preferable page. For example, create a `PageNotFound.htm` page with explanatory text. Then open the `web.config` file and insert the following `<error>` element (it’s in bold) inside the `<customErrors>` element you added previously:
Catching and E-Mailing Exceptions

In an e-commerce site, unhandled exceptions are more than an inconvenience to users and an embarrassment to site owners. Errors can be expensive because when the page that accepts credit card payments fails, the site loses sales. Sending an e-mail to the support desk (or whoever’s minding the store) with a copy of the unhandled exception is useful. That way, someone can start investigating right away.

In ASP.NET, an unhandled exception triggers an event that you can catch in the global.asax file — a type of master code file. Follow these steps to create an e-mail message with the details of a site error:

1. Add a Global Application Class file named Global.asax to your project (File ➪ New File ➪ Global Application Class ➪ Add).
2. In the Global.asax file, locate the Application_Error() subroutine.
   
   As indicated in the handler code comments, Application_Error() is the routine that runs when an unhandled exception occurs.
3. Replace the existing Application_Error() subroutine with Listing 21-1.
4. Change the e-mail addresses, passwords, and mail server details (shown in bold in Listing 21-1) with your Web site and mail server's information.
   
   Check with your ISP’s technical support for the correct values to use in Listing 21-1. If you use goldie@kencox.ca, don’t expect a response. Goldie’s a good dog but she’d rather play ball than answer her e-mail.
5. If you have a <customErrors> element in the web.config file, set the mode="Off".
6. Without using the debugger, browse (Ctrl+F5) to the page that throws an unhandled exception.
   
   The browser redirects to the oops.htm page, and the configured e-mail account receives an error message similar to the one in Figure 21-3.
Listing 21-1: Sending an Unhandled Exception via E-Mail

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    Try
        Dim sb As New StringBuilder
        Dim mailaddr As New System.Net.Mail.MailAddress
            (="goldie@kencox.ca", "Goldie")
        Dim mailer As New System.Net.Mail.SmtpClient()
        Dim creds As New System.Net.NetworkCredential
            (="goldie@kencox.ca", "pwd")
        Dim objErr As Exception
        objErr = Server.GetLastError().GetBaseException()
        Server.ClearError()
        sb.AppendLine("<b>Source:</b>" & Request.RawUrl)
        sb.AppendLine("<br><b>Browser:</b>" & Request.UserAgent)
        sb.AppendLine("hr<br>\ Error in: </b>" & Request.Url.ToString())
        sb.AppendLine("hr\ Error Message: <b></b> & objErr.Message.ToString())
        sb.AppendLine("hr\ Stack Trace: <b></b> & _
            objErr.StackTrace.ToString())
        mailmsg.IsBodyHtml = True
        mailmsg.Subject = "Error from Web site"
        mailmsg.To.Add(="goldie@kencox.ca")
        mailmsg.Body = sb.ToString()
        mailmsg.From = mailaddr
        mailer.Host = ="mail.kencox.ca"
        mailer.UseDefaultCredentials = False
        mailer.Credentials = creds
        mailer.Send(mailmsg)
    Catch exc As Exception
    Finally
        Response.Redirect("~/oops.htm")
    End Try
End Sub
Here are the highlights of Listing 21-1:

→2 Start a Try...Catch...Finally sequence. This is protection against the error handler throwing an exception of its own. See “Using Try...Catch in Risky Situations” later in this chapter.

→4-8 Set up the objects required for sending an e-mail message including the credentials required by the SMTP server.

→9-10 Create an Exception object and put the most recent exception (the one that caused this code to execute) into the object.

→11 Call the Server object’s ClearError() method to ensure that the same exception doesn’t get passed to the web.config file.

→12-14 Start assembling the contents of the e-mail message. This part uses the built-in Request object to find out details such as the type of browser and the URL that caused the exception.

→15-17 Add details of the error to the e-mail message including the actual error message (“Could not find a part of the path ‘c:\doesntexist’”) and the report found in the StackTrace. The StackTrace property pinpoints the throw point where the exception occurred such as “at System.IO.__Error.WinIOError(Int32 errorCode, String maybeFullPath)”. This error text can be many lines long.

→18-22 Configure the e-mail message by adding the subject, recipient, sender, and the body of the message.

→23-26 Configure the SmtpClient object and use its Send() method to send the e-mail message.

→27 Catch any error thrown during this error handler routine but don’t do anything with the error (eat the error in geekspeak).

→28-29 No matter what else happens, send the browser to the polite “Oops” page so the visitor never sees us sweat.

Exceptions aren’t always forthcoming about what went wrong. Look inside an Exception object to find its InnerException property to discover what’s really bothering the program.

**Using Try...Catch in Risky Situations**

Sometimes you sense that your code is getting into a risky situation — usually when your routines depend on something external, such as the existence of a file, a directory, or data connection. Instead of relying on a global error handler, deal with the exception locally.
In this example, you use a `Try...Catch` block to wrap code that might throw an exception. Additionally, you’re quite specific about the exceptions that you’re willing to handle locally. **Note:** This example assumes that you are not using the `Global.asax` file to handle application errors.

To catch and report specific exceptions, follow these steps:

1. **Add an ASP.NET GridView control to a page.**
2. **Add a Label control with an ID of `lblError` to the page.**
   As shown in Figure 21-4, the Label provides feedback to the user in case something goes wrong.

3. **In Source view, add `<%@ Import Namespace="System.Linq" %> to the top of the .aspx page for LINQ query support.**
4. **In Source view, use Listing 21-2 as the handler for the Page object’s Load event.**

**Listing 21-2: Catching and Reporting Specific Exceptions**

```csharp
Protected Sub Page_Load (ByVal sender As Object, ByVal e As System.EventArgs)
    Try
        Dim q = From FileName In _
            System.IO.Directory.GetFiles _
            ("c:\doesntexist", "*.*", _
                System.IO.SearchOption.AllDirectories)

        GridView1.DataSource = q
        GridView1.DataBind()

    Catch ex As System.Security.SecurityException
        lblError.Text = "Sorry, you don’t have permission for this."
    Catch ex As System.IO.DirectoryNotFoundException
        lblError.Text = "Sorry, couldn’t find the directory."
    End Try
End Sub
```
Catch ex As Exception
    lblError.Text = "Please forward this to the admin:<br>" & ex.Message
End Try
End Sub

Assuming that you don’t have a doesntexist folder on your c:\ drive, the runtime message in the browser reads Sorry, couldn’t find the directory.

Here are the parts of the Try...Catch...End Try blocks:

→4 The opening Try statement for the structured exception handling announces, “Lookout! Something in the code that follows could cause a problem.”

→13 Catch ex As System.Security.SecurityException — the first Catch — is looking for a specific exception dealing with security. The account that runs ASP.NET commonly doesn’t have sufficient permissions to access the file system.

→14 If the SecurityException was a match, the code steps into the next statement which tells you the problem by writing a bit of text to the Label control’s Text property: lblError.Text = "Sorry,...".

→16 Catch ex As System.IO.DirectoryNotFoundException — the second Catch — is also looking for a specific exception, DirectoryNotFoundException. The test for a missing directory might never happen because it’s second in line. If the SecurityException was thrown, the handler exits and the DirectoryNotFoundException becomes irrelevant.

→17 Similar to the previous assignment, you’re informed about the directory problem when the DirectoryNotFoundException is thrown.

→19 Catch ex As Exception is the fallback handler, which executes only when the preceding specific exceptions have no matches. Nothing gets past this generic catchall exception, which is why it comes last.

→20 If lblError.Text = "Please ...<br>" & ex.Message executes, you have no idea what went wrong. Instead of giving you a friendly but useless explanation, the programmer passes the error text that’s contained in the Exception object’s Message property. It’s a judgment or policy call whether to gloss over the error or show users what actually happened.

→21 End Try signals the end of the error handler that started with Try.
Another block in the Try...End Try sequence, Finally, is handy for cleaning up operations. For example, if the code opens a data connection successfully but another statement throws an exception, Finally can tidy up and close the connection. It’s not a full recovery; however, it reduces the exception’s collateral damage.

In Listing 21-3, the Try block includes a Throw statement to cause an error programmatically (and to illustrate how to create and raise custom exceptions). A Catch statement catches the deliberate exception immediately and puts the error object into its own variable, ex. The lblErr.Text = ex.Message line displays the error message on-screen.

To show that the Finally statement is executing, the code inside the Finally block adds its text to the existing message.

Listing 21-3: Throwing, Catching, and Finally

```vbnet
Protected Sub Page_Load (ByVal sender As Object, ByVal e As System.EventArgs)
    Try
        Throw (New Exception("A custom exception!"))
    Catch ex As Exception
        lblErr.Text = ex.Message
    Finally
        lblErr.Text = lblErr.Text & "<br>Finally executed."
    End Try
End Sub
```
People like to say that `Finally` executes, no matter what. This list of conditions shows that’s not quite true:

- **Try** block runs without error: `Finally` block executes.
- Exception thrown in **Try** block and caught in a **Catch** block: `Finally` block executes.
- Exception thrown in **Try** block, caught in **Catch** block, and rethrown in **Catch** block: `Finally` block doesn’t execute.
- Exception thrown in **Try** block and uncaught in **Catch** block(s): Too bad, but `Finally` block doesn’t execute. The unhandled exception goes somewhere else for processing (*farther up the stack* in geekspeak).

### Some Common Error Messages and Where to Look

Runtime errors that happen often enough find their way onto Frequently Asked Question (FAQ) lists. In this section, I show you some exceptions that you’re likely to encounter and provide you some guidance on resolving them.


Your pages are running fine on your computer but fail when you post them to the Web site on the Internet. The exception complains: *The application attempted to perform an operation not allowed by the security policy.*

Web hosting companies usually lock the trust level of sites to medium rather than full. The following setting (at the machine-level `web.config` file) overrides whatever you try to set in your Web site:

```xml
<trust level="Medium" originUrl="" />
```

If a page or called component tries to use the file system (usually outside your Web site), Registry, event log, OLEDB data access, or restricted assem- blies, the constraint throws a `SecurityException` that crashes the page.

Ask the Web host to review the trust level and perhaps allow an override or exception. To locate information on appropriate settings, search Microsoft’s Web site for *How to use medium trust in ASP.NET*. If you’re using a third-party component, check with the vendor whether their component requires full trust and if so, what you can do to make it work under medium trust.
System.NullReferenceException

People ask about this one all over the Internet: Object reference not set to an instance of an object. You’re trying to do something with a variable that has the value Nothing or null. Chances are, you forgot to use the keyword New to create an instance of the object you’re trying to use. Another possibility is that the object has disappeared. This happens when you use an object that was stored in a Session variable but the Session timed out.

If you’re using a String type, you don’t need New; however, assign a value to the variable right away, even if it’s an empty string. The following version causes a runtime error, so don’t use it in your code:

```vbnet
Dim strText As String
Label1.Text = strText.Length.ToString
```

The following code doesn’t cause an error because of the assignment of an empty string:

```vbnet
Dim strText As String = ""
Label1.Text = strText.Length.ToString
```

Bizarre as it seems, an empty string (" ") counts as something and avoids the problem of string variables that are Nothing.

If you’re getting the error while trying to display data on a Web page, check that you actually received something when your code fetched the DataTable or DataSet object.

Are you missing an assembly reference?

This error occurs when you’re using objects in external assemblies (DLLs), such as third-party libraries and components. The error reads

```
The type or namespace name '<some name here>' does not exist in the namespace '<another name here>' (are you missing an assembly reference?)
```

If you’re getting this error in your Visual Web Developer project, try these steps to add a reference:

1. In the VWD project, ensure that the DLL your code calls is available in the bin folder.
2. In Solution Explorer, right-click the project name (the root) and choose Property Pages from the context menu that appears.
3. On the left side of the Property Pages window, click References and then click the Add button (lower right).

4. Select the component name or Browse to the assembly and then click OK.

**'Button1_Click' is not a member of 'ASP.default2_aspx'**

Everything is working fine until suddenly you get an error referring to an event handler in your page. Here's a sample error:

```
Compiler Error Message: BC30456: 'Button1_Click' is not a member of 'ASP.default2_aspx'.
```

The `Button1_Click` part and the `default2_aspx` part change according to the name of the control and the name of the ASP.NET page.

The problem is usually that you inadvertently double-clicked an ASP.NET control and VWD created a default event handler for you. You probably noticed this happening (because the editor switched to Source view) and therefore you deleted the skeleton handler subroutine. However, VWD also changed the control’s markup by inserting a reference to the (now-deleted) subroutine. Here’s the troublesome part that you missed:

```
onclick="Button1_Click"
```

The fix (always simple after you find it!) is to delete the event handler reference from the control’s markup.

**Expression of type ‘1-dimensional array’ is not queryable**

You’re probably using LINQ when you see the “not queryable” message. The usual fix is to do what the error message suggests. Add this to the top of your .aspx page:

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

In regular code, it looks like this:

```
Imports System.Linq
```

Or, to add an assembly reference, right-click the project name and select Add Reference. On the .NET tab, select System.Data.Linq and click OK.
Part VI

The Part of Tens

The 5th Wave

By Rich Tennant
In this part...

The Part of Tens. Don’t ask me what it means or what oddball tradition inspired it! I’m just a book author told to provide a Part of Tens. It’s a happy coincidence that Chapter 22 fits the pattern with ten solid tips on getting your content onto a Web server.

Chapter 23 didn’t work out so well. I originally wrote 11 anti-head-bashing tips for Chapter 23. One of them got the chop due to the Part of Tens thing. Because my editor doesn’t read these rambling introductions, I’m going to try to disguise the eleventh tip and sneak it into the next paragraph.

While you work through the final chapter in this part, keep in mind that if you’re really stuck on something you’ve read in the book, contact me, Ken Cox, at kjopc@hotmail.com or visit my site at www.kencox.ca. There. Everybody’s happy.
Chapter 22

Ten Tips on Deploying Your Web Application

In This Chapter

- Using the Copy Web Site tool
- Publishing SQL Server content
- Fighting with SQL Server connections
- Dealing with a lack of (full) trust
- Precompiling and encrypting

There you sit, day after day, toiling away at the greatest ASP.NET pages the world has never seen. Finally, it’s time to put something on the Internet for a test, a proof-of-concept, or maybe production use. Deployment — at least the first time — is a nuisance and a pain. This chapter offers tips that might get you through the worst of it.

Use the Copy Web Site Tool

The easiest way to transfer files between your workstation and the remote Web server is the Copy Web Site tool, as shown in Figure 22-1. The tool connects to the remote site by using the file system, the FTP protocol, and the FrontPage Server Extensions for Internet Information Services (IIS).
Connecting via FTP

File Transfer Protocol (FTP) is the most common way to deploy ASP.NET files to a Web site on the Internet. The hosting service provides the username, password, and FTP server name that you require. Follow these steps to connect by using FTP:

1. In Visual Web Developer, open the Web project that you want to deploy.
2. Choose Website ➪ Copy Web Site.
   The Copy Web tool appears. (Refer to Figure 22-1.)
3. Click the Connect button (near the top of the tool area).
   The Open Web Site dialog box appears, as shown in Figure 22-2.
4. In the left panel, click FTP Site.
5. In the Server box, enter the name of the FTP server, for example, ftp.kjopc.com.
6. Type the credentials (username and password) in the corresponding boxes.
7. Click Open.

If the remote storage folder is below the root of the FTP site, save time by entering the path to the folder in the Directory box.
**Connecting by using the FrontPage extensions**

*FrontPage Server Extensions* (FPE) is an add-on for Internet Information Services that allows you to log in and manage a Web site. FPE isn’t installed by default on the Web server, so you need to check its availability with the hosting company or your Web server administrator and find out the credentials.

1. In Visual Web Developer, open the Web project that you want to deploy.
2. Choose Website ➪ Copy Web Site.

   The Copy Web tool appears. (Refer to Figure 22-1.)
3. Click the Connect button (near the top of the tool area).

   The Open Web Site dialog box appears
4. In the left panel, click Remote Site (see Figure 22-3).
5. In the Web Site Location box, enter the URL or IP address of the remote site.
6. Click Open.
Connecting via the file system

Connecting via the file system is usually restricted to intranet situations where the Web server is on the same network and the administrator has shared its folders for your use. On my home network, I have an intranet Web server (IISERVER). I share the `c:\inetpub\wwwroot\aspnet35fd\` folder by using the share name `aspnet35fd`. To connect via the file system, follow these steps:

1. In Visual Web Developer, open the Web project that you want to deploy.

2. Choose Website ➪ Copy Web Site.

   The Copy Web tool appears. (Refer to Figure 22-1.)

3. Click the Connect button (near the top of the tool area).

   The Open Web Site dialog box appears.

4. In the left panel, click File System and then navigate to the network location and shared folder name.

5. If prompted, supply the username and password to the shared folder.

   Figure 22-4 shows a sample folder named `\IISERVER\aspnet35fd`.

6. Click Open.
Transferring files in the Copy Web tool

After you connect to the remote location, the Copy Web tool shows the source Web site on the left and the remote Web site on the right. To copy files from the source to the remote site, follow these steps:

1. In the left pane, select the source site files and folders that you want to copy to the remote site. (Shift+click to multiselect.)
2. Click the blue, right-facing arrow located between the two directory panes.

The Status area at the bottom of the Copy Web tool shows the latest operation. If you’re overwriting files, the tool prompts you to confirm.

Tip
If you need to fetch files from a remote site that you haven’t worked on before, create a file system Web site on your development machine, connect to the remote machine, select all the remote files and folders, click the blue, left-facing arrow, and then take a coffee break while the files transfer. (Copying can be slow.)

Use the SQL Publishing Wizard

Visual Web Developer (VWD) includes a SQL Publishing Wizard that helps you deploy development database content to the remote SQL Server. Hosting companies use different schemes for managing these databases, so it’s
impossible to give exact instructions for the host. For example, you may need to create the database manually in a graphical tool before proceeding with the script to generate and populate the tables.

Creating a database script

A SQL script that creates the database tables and inserts the data is commonly required. The SQL Publishing Wizard generates the .sql script based on your local database. Follow these steps to create the SQL script for an existing database:

1. Open a project that uses a SQL Server (including Express) database.
2. Open Database Explorer (Server Explorer in non-Express versions).
3. After ensuring that the connection is open, right-click the data connection and choose Publish to Provider from the context menu that appears.

The Database Publishing Wizard opens.

If the wizard fails, return to Database/Server Explorer and expand the data connection node to force the connection state to Open and try again.

4. Click Next.
5. In the Select Database window, select the database that you want to publish and then click Next.
6. In the Select an Output Location window, select the Script to File option, enter the location and name of the script file to write (for example, c:\temp\juliedatabase.sql), and then click Next.

The Select Publishing Options window appears.

7. As shown in Figure 22-5, select the types of data to publish and then click Next. The following table helps you choose an option.

<table>
<thead>
<tr>
<th>Option</th>
<th>Scenario</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop existing objects in script</td>
<td>Set to True if you want to remove (drop) the existing tables and stored procedures from the remote database and replace them with new versions from the local database.</td>
</tr>
<tr>
<td>Schema qualify</td>
<td>Leave as True to qualify object names with their schema name (this generates two-part names).</td>
</tr>
<tr>
<td>Script for target database</td>
<td>Select the version of SQL Server the remote database uses.</td>
</tr>
</tbody>
</table>
Option | Scenario
--- | ---
Types of data to publish - Data only | You have existing tables in the remote database and want to add only the data stored in the local database.
Types of data to publish - Schema and data | You want the remote database to have the same tables, structure, and data as the local database.
Types of data to publish - Schema only | You want the remote database to have the same tables and structure as the local database but don’t want to transfer the local data.

8. In the Review Summary window, click Finish and, when the process is complete, click Close.

---

**Creating a remote database from a script**

After the SQL script is created by the SQL Publishing Wizard, connect to the remote SQL Server and run the script. How you execute the script depends on which tools your hosting service supports. If you have rights to log on directly to the remote SQL Server, use SQL Server Management Studio Express. Search the Microsoft site for *SQL Server Management Studio Express* for the free download. To execute the script, follow these steps:

1. Log in to SQL Server with SQL Server Management Studio (Express), as shown in Figure 22-6.
2. In Object Explorer, select the SQL Server instance to create the database, as shown in Figure 22-7.

3. Choose File➪Open. In the Open File dialog box, navigate to the script that you created with the Publishing Wizard and click Open.

The content of the script appears in the script window.

4. Choose Query➪Execute.

The Management Studio starts executing the query. A long script can take several minutes to run.

Some providers use a browser-based SQL Web Admin interface. In that case, use its Query Analyzer or similar tool that runs scripts. In Notepad, open the SQL script that the wizard created, copy the text to the clipboard, and then paste the script contents into the hosting provider’s database query interface.

**Copy a SQL Express Database**

Some hosting providers support SQL Express and detached databases. This allows you to disconnect from your local machine, detach the database, copy the .mdf file to the host site via the Copy Web Site tool, and attach the database to the host’s SQL Server or SQL Express.
See the hosting company’s instructions for the specifics. A Web-based wizard might be available to walk you through the upload and connection process.

**Fix the @#$%*& SQL Connection**

Connecting ASP.NET to a remote SQL Server or SQL Server Express database is error-prone and frustrating. If you manage to connect the first time without any grief, run out and buy a lottery ticket — you’re on a roll!

The connection string usually resides in the `web.config` file in the `<connectionStrings>` section. You need to tweak this while trying to connect deployed pages to the data. Here’s a typical SQL Server Express connection string as used within a local project:

```xml
<add name="JulieDVDConnectionString1" connectionString="Data Source=.;AttachDbFilename=|DataDirectory|JulieDVD.mdf;Integrated Security=True;User Instance=True" providerName="System.Data.SqlClient"/>
```

The preceding uses SQL Express, an attached file, and Windows authentication. Odds are, those settings make no sense to your host service.

The same connection — revised to use the host service’s SQL Server — might look similar to this:

```xml
<add name="JulieDVDConnectionString1" connectionString="Data Source=SQLB2.webcontrolcenter.com;Initial Catalog=juliedvd;Persist Security Info=True;User ID=julie;Password=nottellingyou!" providerName="System.Data.SqlClient"/>
```

The preceding connection string uses the URL (`SQLB2.webcontrolcenter.com`) of a SQL Server on the Internet as its data source. The connection string also uses SQL authentication by providing a user ID and password known to the database. Connection details are often included in the host service’s welcoming e-mail message.

Use a browser-based utility to add and configure database connections for your site.

Look for a frequently asked questions (FAQ) page on database connections. You’re not the first to run into this hassle, so the hosting service usually posts instructions that are specific to their setup.
Choose an ASP.NET-Friendly Host

When you create Web sites in ASP.NET, you almost always need a server that runs Windows Server 2003 or newer. I say *almost* because some hosting services might offer Linux machines that run a version of ASP.NET under Mono. For best results, find a hosting service that has experience with your version of ASP.NET. Better yet, find one that participated in Microsoft’s ASP.NET beta testing.

Head Off a Serious Lack of Trust

Hundreds of companies sell precompiled assemblies to enhance your Web pages, perform calculations, generate documents, and work miracles with .NET. Most vendors provide trial versions to play with before spending your (or your company’s) cash. If the component works fine on your development system, check with the vendor on the exact permissions the component requires on the Web server. Specifically, ask whether it runs under Partial Trust.

Web-hosting companies set rules on their systems for ASP.NET and its components. For shared hosting, most companies restrict ASP.NET to running under Partial Trust. In the Partial Trust scenario, Web pages usually aren’t allowed to access the Windows Registry or perform file access operations outside the site’s directory. If any component that ASP.NET uses requires Full Trust — or depends on something that requires Full Trust — the permissions break down. Workarounds, such as using a virtual server or even a dedicated machine, are more expensive than shared hosting.

A component that runs perfectly in the freewheeling Visual Web Developer environment might halt in the restrictive Partial Trust world.

Arrggh! It Works Fine on MY Machine!

Here’s where deployment can drive you mad. Pages run fine on your development computer but then crash when deployed, leaving you to find what’s different about the destination (target) site. Here are some issues to look for:

- **Required assemblies are stored in the Global Assembly Cache (GAC) on your machine but not on the destination machine.** Because these assemblies don’t appear in your `bin` folder, you may not realize that your code depends on them. Contact the hosting provider with details or, as a last resort, copy the assemblies to your site’s `bin` folder.
Tighter permission rules exist on the destination machine, or ASP.NET is running an account with fewer privileges. Change the permissions (such as Read/Write/Execute) on your destination machine's folders. See the earlier section, “Head Off a Serious Lack of Trust.”

The destination machine doesn’t have the identical version of the .NET runtimes. Make sure the hosting provider has upgraded to the latest service pack.

The Web server is using a different culture setting and is misinterpreting dates. Try running the script in the upcoming section, “Gather Troubleshooting Info.”

Hardcoded paths to resources, such as images, exist. These paths (for example, c:\inetpub\wwwroot\mysite\images\yikes.gif) make sense on your development computer but are lost on the destination machine. Paths for ASP.NET controls should be relative and start with the tilde character (~), which stands for the site’s root.

Third-party components don’t have the correct license for the domain. Some vendors allow you to develop with their components on your local machine (localhost) but tie license keys to a domain name or subdirectory name. Contact the vendor.

Gather Troubleshooting Info

If your ASP.NET pages don’t run at all, this section isn’t going to help you. However, if some deployed pages fail and others don’t, Listing 22-1 might reveal the discrepancy. It displays the version of ASP.NET, the server name, the account that ASP.NET is running as, the file system root of the Web, and the culture setting.

Listing 22-1: A Page to Gather Troubleshooting Info

```vbnet
<%@ Page Language="VB" Trace="true" %>
<script runat="server">
    Private Sub Page_Load( ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Try
            Dim ver As System.Version
            ver = System.Environment.Version
            Response.Write("Version: " & ver.ToString() & "<br />")
        Catch exc As Exception
            Response.Write(exc.Message & "<br />")
        End Try
    Try
        Try

(continued)```
Listing 22-1 (continued)

```vbnet
Dim wi As System.Security.Principal.WindowsIdentity = _
Response.Write("Account: " & wi.Name & "<br />
Catch exc As Exception
   Response.Write(exc.Message & "<br />
End Try
Response.Write("Web root: " & Server.MapPath("~") & "<br />
Response.Write("Server name: " & _
   Request.ServerVariables("SERVER_NAME") & "<br />
Response.Write( _
      CurrentCulture.Name & ": " & _
      CurrentCulture.EnglishName & "<br />
End Sub
</script>
```

**Precompile If You’re Code Shy**

If it bothers you that someone might rummage around on the Web server and read your source code, consider precompiling the site. Normally, ASP.NET compiles the code the first time someone requests a page. With precompilation, ASP.NET puts the code from the .aspx, .asmx, .ascx, and .vb files into a few assemblies (.dll files) in the bin folder. Follow these steps to compile your Web project:

1. **Open your project in Visual Web Developer and build the site (Build ➔ Build Web Site) to see any errors or warnings that need attention (View ➔ Error List).**
   
   Precompilation fails if the project has errors, so fix or exclude troublesome pages.

2. **Open a command prompt in the directory where aspnet_compiler.exe is installed. (Try C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727.)**

3. **At the command prompt, enter the following command, replacing c:\site\kjopc with the path to your Web files:**

   ```
   aspnet_compiler -v /myweb -p c:\site\kjopc c:\deploy
   ```

   The preceding command compiles and then copies the required files (including images and other static content) into a new c:\deploy folder. The compiled files appear in the bin folder.
Although the .aspx files copy to the `c:\deploy` folder, no code or markup is in them — just the warning text: This is a marker file generated by the precompilation tool, and should not be deleted!

Deploy the contents of the `c:\deploy` folder knowing that your code and markup are stored in binary (compiled) files. Without adding obfuscation, probing and recovering the source code from .NET assemblies is quite easy with decompilation tools, such as Reflector.

**Encrypt Connection Information**

ASP.NET doesn’t *serve* (display) certain types of files to the browser, including configuration files. However, don’t tempt fate by flashing a SQL Server password in the `web.config` file. Follow these steps to encrypt the connection strings in the `web.config` file:

1. **Open a command prompt and change to the directory where `aspnet_regiis.exe` is stored.** (Try `C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727`.)

2. **Enter the following command, replacing "c:\deploy" with your deployment folder:**

   ```
   aspnet_regiis -pef "connectionStrings" "c:\deploy"
   ```

   The tool adds several nodes to the `<connectionStrings>` element, such as `<EncryptedData>` and `<CipherValue>`. Your secrets are somewhere within all those nonsense characters and harder to decipher than the DaVinci code — except by ASP.NET!

   **This command decrypts the section:**

   ```
   aspnet_regiis -pdf "connectionStrings" "c:\deploy"
   ```
Chapter 23

Ten Tips to Success with ASP.NET

In This Chapter
- Why you shouldn’t feel dumb
- Helping yourself with a little research
- Asking questions and getting answers
- ASP.NET community resources
- Filling your toolkit for free

A huge, friendly user community supports ASP.NET. Volunteers create helpful Web sites, tools, and tutorials. Members, such as MVPs, answer questions in newsgroups and Web forums. Developers offer valuable sample applications that you can download, use, and absorb. In this chapter, I point you toward a wealth of free resources that further your success with ASP.NET.

Stop Bashing Your Head against a Wall

It’s not unusual to be stuck on code that should work but doesn’t. It happens to everyone, not just beginners. When you’re spinning (geekspeak for getting nowhere) the frustration can drive you up the wall. Consider these possibilities when you hit a roadblock:

✔ There’s a knowledge gap. You’re missing a key piece of information that you need to make this thing work.

✔ The code sample is wrong. It might be that the code was based on a prerelease version and hasn’t been updated; that it targets a different version of .NET; that it never worked in the first place; or something has been left out.

✔ The documentation is wrong. Because of product churn (last minute and unreported changes), the documentation may not accurately reflect the release version of the software. (Churn is an occupational hazard for technical writers!)
You’re up against a bug in the .NET platform or ASP.NET. Released software always has some bugs, so it’s within the realm of possibility that you discovered one. That said, be sure you rule out the preceding possibilities before deciding the platform is faulty.

The great thing about ASP.NET is that you’re never alone. A good chance exists that someone has already jumped the same hurdle.

Google Is Your Friend

A quick search using Google (www.google.com) or Microsoft’s Live Search (search.live.com) often provides a solution, especially if you include the exact error message. Sometimes you need to enter only a question about what you want to know, such as this query:

how to create a hierarchical gridview in asp.net

Read the Reference Documentation

The .NET Framework Class Library (FCL) is so vast that it requires a searchable, electronic retrieval system to make it usable. Microsoft publishes complete documentation on the classes, interfaces, and types required to create ASP.NET and other .NET applications.

Built-in online help

Visual Web Developer includes a built-in online Help engine and gobs of valuable information. Within the IDE, press the F1 key to launch the help viewer or use the Help menu items.

Web-based reference material

Microsoft’s Web-based MSDN Library has the most up-to-date reference material, including information added by community members. (If your knowledge can save someone pain, add the info to the official documentation.)

Because URLs change, your best bet is to navigate to http://search.msdn.microsoft.com/ and search for .NET Framework Class Library.
Ensure that you’re reading the documentation for the latest version of .NET. For .NET 3.5, the URL usually includes the version vs.90 in parenthesis, such as http://msdn2.microsoft.com/en-us/library/ms229335 (vs.90).aspx.

Ask a Good Question, Get a Good Answer

After you search for an answer and try to resolve the problem, don’t be shy about asking for help in the ASP.NET newsgroups and Web forums. Don’t worry about looking like a dummy — nobody knows everything and everyone starts with zero expertise.

To get a good answer for a coding issue, boil your problem down to a simple example. If someone can run your code and reproduce the error without going to a lot of trouble, you stand a good chance of getting a solution quickly. Reporting, “It doesn’t work,” isn’t enough. Include the error message and a description of what’s failing along with what you want or expect to happen.

Get Free Peer-to-Peer Support

Microsoft sponsors Web-based forums where community members, such as Microsoft Most Valuable Professionals (MVPs) and Microsoft employees, provide free support and advice on ASP.NET, Visual Web Developer, and Visual Studio.

Join forums.asp.net

To reach the official ASP.NET forums, navigate your browser to this site:

http://forums.asp.net/

You can peruse the site anonymously, but create an account and a profile if you want to post questions.

Enable the E-mail Subscription feature on the forums. The forum software sends an e-mail message whenever someone replies to your question. That way, you don’t need to remember to revisit the forums to check for answers.

For those who prefer to use a specialized newsgroup reader (such as Outlook Express) on the ASP.NET forums, the setup instructions are here: http://forums.asp.net/nntp.aspx.
Find experts at msnews.microsoft.com

Many top ASP.NET experts hang out in Network News Transfer Protocol (NNTP) newsgroups rather than browser-based forums. (Power users often find Web forums slow and clunky.) To get to the action, you need a newsreader, such as Outlook Express. Follow these steps to subscribe to microsoft.public.dotnet.framework.aspnet.

1. Launch Outlook Express.
2. Choose Tools ➔ Accounts.
3. In the Internet Accounts window, click Add and then click News.
   The Internet Connection Wizard opens.
4. In the Display Name box, enter your name and then click Next.
5. In the E-Mail Address box, enter a spammer resistant version of your e-mail address, such as BANSPAMmyname@domain.com, and then click Next.
6. In the News (NNTP) Server box, enter msnews.microsoft.com and then click Next.
7. After Outlook Express downloads the available newsgroups, select microsoft.public.dotnet.framework.aspnet, click Subscribe, and then click OK.

Use the Starter Kits

Starter kits are free applications that include documented source code, style sheets, and databases. Before you tackle a personal Web site, club site, classified ads application, e-commerce page, catalog, wiki, blog — any major application — browse the list of starter kits. Here’s where to start:

www.asp.net/community/projects

After using a few starter kits, you might agree with the old-timers who say, “All the code has already been written.”

Read the Hottest Blogs

Undoubtedly, if an issue’s burning about ASP.NET, someone is discussing it on his blog. Whether it’s a workaround to a nasty bug, the introduction of a new technology, or a Visual Web Developer add-on, blogs are up-to-the-minute sources. Be sure to visit http://weblogs.asp.net/ where members of
Microsoft’s ASP.NET team and many top community members appear. Above all, read master-blogger and ASP.NET boss, Scott Guthrie (http://weblogs.asp.net/scottgu/). Scott’s tips, tricks, and tutorials merit appreciative reviews in his comments section.

Another major technoblog site to visit is http://blogs.msdn.com/. Dozens of Microsoft employees write about their technology areas and post product announcements. Be sure to watch Somasegar’s WebLog (http://blogs.msdn.com/somasegar/) for news about Visual Web Developer and Visual Studio from the guy in charge of the division.

**Watch the Videos**

Although many people learn best by doing, you can also pick up many techniques by watching video tutorials. An excellent collection is at http://www.asp.net/learn/.

**Visit the Expert Web Sites**

Members of the ASP.NET community have created dozens of Web sites that include news, tips, and full articles on Web site programming. Here’s a list of some of the more popular free sites. You find links to even more content while browsing their pages.

<table>
<thead>
<tr>
<th>Site</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GuysFromRolla</td>
<td><a href="http://aspnet.4guysfromrolla.com">http://aspnet.4guysfromrolla.com</a></td>
</tr>
<tr>
<td>ASP Alliance</td>
<td><a href="http://aspalliance.com">http://aspalliance.com</a></td>
</tr>
<tr>
<td>ASP Free</td>
<td><a href="http://www.aspfree.com">www.aspfree.com</a></td>
</tr>
<tr>
<td>Code Project</td>
<td><a href="http://www.codeproject.com">www.codeproject.com</a></td>
</tr>
<tr>
<td>CodePlex</td>
<td><a href="http://www.codeplex.com">www.codeplex.com</a></td>
</tr>
<tr>
<td>CSharp Friends</td>
<td><a href="http://www.csharpfriends.com">www.csharpfriends.com</a></td>
</tr>
<tr>
<td>DotNetBips</td>
<td><a href="http://www.dotnetbips.com">www.dotnetbips.com</a></td>
</tr>
<tr>
<td>Dotnetjunkies</td>
<td><a href="http://www.dotnetjunkies.com">www.dotnetjunkies.com</a></td>
</tr>
<tr>
<td>DotNetSlackers</td>
<td><a href="http://dotnetslackers.com">http://dotnetslackers.com</a></td>
</tr>
<tr>
<td>EggHeadCafe</td>
<td><a href="http://www.eggheadcafe.com">www.eggheadcafe.com</a></td>
</tr>
<tr>
<td>SingingEels</td>
<td><a href="http://www.singingeels.com">www.singingeels.com</a></td>
</tr>
</tbody>
</table>
Use the Free Tools

To dig deeper into the HTML and script that your ASP.NET application is exchanging with a browser, get “close to the wire.” Here are some advanced tools that I use to get an X-ray view of the application’s innards:

- **Fiddler2**: This program logs all HTTP(S) traffic between your computer and the Internet so that you can view HTTP headers, text, cookies, and URLs. Additionally, you can adjust and replay the content of Web forms to see how the Web server responds. It’s free at www.fiddler2.com.

- **Internet Explorer Developer Toolbar**: A great plug-in for analyzing the hierarchy of tags in a Web page. You can select an object, such as a table, and view the properties, styles, and markup that it contains. To download a copy, navigate to www.microsoft.com/downloads and search for Internet Explorer Developer Toolbar.


- **Web Development Helper**: A project by Microsoft’s ASP.NET guru, Nikhil Kothari, this Internet Explorer plug-in allows you to trace and log interactions between the browser and the Web server. You can view the current values of objects as Internet Explorer sees them by poking about in the Document Object Model (DOM). Download the plug-in from http://projects.nikhilk.net/Projects/WebDevHelper.aspx.
Symbols

. (dot)
  class-based style rules, 200
  object initialization, 116
* (asterisk), wildcard character, 105
= (equal sign), object initialization, 116
!! (exclamation marks), extension methods, 138
_ (underscore), line break character, 101

A

<a> (anchor tag), 223–225
accelerator keys. See access keys
Access databases, 151–152
access keys, 205–206
accessibility, 203–204
AccessKey property, 206
adding (arithmetic). See summing
adding (inserting). See inserting
add-ons. See extensions
Admin folder, 285–288
administration area, 284–286
administrator role, 276
administrators, adding, 288–289
ADO.NET, 19
aggregating data
  averaging returned values, 108–109
  counting returned items, 107–108
  displaying in chunks, 110–111
  minimum/maximum values, 109–110
  summing returned values, 109
AJAX, description, 15–16
AJAX Control Toolkit
definition, 239
downloading, 31
extenders
  always on top, 251–254
AlwaysVisibleControlExtender, 251–254
AutoCompleteExtender, 241–244
auto-completion feature, 241–244
CalendarExtender, 249–251
date-picker, 249–251
floating text over pages, 251–254
listing, 240–241
MaskedEditExtender, 247–248
masking text input, 246–248
performance improvement, 60
refreshing Web pages, 60
text box prompts, 244–248
TextBoxWatermarkExtender, 245–246
watermarks, 244–246
installing, 31–32
alternating visual appearance, 216
AlternatingItemTemplate, 216
always on top, 251–254
AlwaysVisibleControlExtender, 251–254
anchor tag (<a>), 223–225
animation. See rich content
anonymity, 305
anonymous types, 101
anonymous visitors, 275
The application...operation not allowed... message, 359
arrays, 107–108, 157
ASP.NET 3.5, 12–13
ASP.NET 3.5 Extensions, 13–14, 260
ASP.NET AJAX. See AJAX
ASP.NET Futures, 13, 260
assemblies, 71
asterisk (*), wildcard character, 105
audio. See rich content
authentication, 273–274. See also
  Membership
authorization, 273–274
Auto Hide, 30–31
AutoCompleteExtender
  data lookup page, creating, 243–244
  data lookup Web service, creating, 242–243
  preparing the word list, 241–242
auto-completion feature, 241–244
AutoFormat feature, 49, 166
Autos pane, 344
Average( ) function, 108–109
averaging returned values, 108–109

• B •
  base classes, 300
  Bind( ) method, 210–211
  binding, 157
  blogs, troubleshooting aid, 382–383
  bold text, 63. See also rich content
Boolean logic, 235
breadcrumbs, 190–191
breaking on conditions, 342–343
breakpoints, 337–339
built-in styles, 166
buttons, 206, 306–308. See also controls;
  radio buttons

• C •
caching expensive content, 201–202
calculations, shopping carts, 311–313
Calculations class, 311–312. See also shopping cart code
CalendarExtender, 249–251
CartItem class. See also shopping cart code
cart items, creating, 298–299
creating, 296–297
declaring variables, 297
decorating, 299
serializing, 299
set/get accessors, 297–298
ColorTranslator, 71
declaring variables, 297
decorating, 299
definition, 71, 295
disambiguating, 295
inheritance, 300
parameters, 298
ProfileCommon, 305
properties, 295
serializing, 299
set/get accessors, 297–298
ShoppingCart, 299–304. See also shopping cart code
XML, 115–117
client scripts, accessibility guidelines, 204
code-behind model, 44
collection editor, 65–66
collections. See .NET collections
colors
  GridView control, 49
  names, converting to types, 71
  selecting from a list, 71
ColorTranslator class, 71
columns (database)
  ID (identity), 39–41
  regular, 41
  sorting, 50, 51
columns (Web page)
dedicated style rules, 199–200
<div> tags, 198–199
float: left style rule, 199–200
overview, 196–198
previewing, 199
command attributes, 83
CompareValidator control, 319–322
comparison operators, 106
compile-time errors, 336–337
connecting to
databases. See also LINQDataSource
control; SqlDataSource control
Access, 151–152
LINQ to SQL, 123
Northwind database, 79–80
SQLExpress, 77–80
RSS feeds, 144
WCF (Windows Communication
Foundation), 160–162
Web services, 155–156
ConnectionString attribute, 82
connectionStrings section, 83
consuming data
Access databases, 151–152
DetailsView control, 84–86
RSS feeds, 143–148
Web references, 155–156
Web services, 152–157. See also WCF
XML, transforming to HTML, 148–151
Contains( ) function, 105
collection editors, 10–11
contracts, 157
controls. See also buttons; specific controls
adding to the Toolbox, 31–32
changing on mouse hover. See rollover
effects
customizing. See templated controls
embedding in templates, 207
formatting, 49. See also templated
controls
grouping, 326
inserting into Web pages, 28
quotes, around inline statements, 209
skins, 176
Copy Web Site tool
description, 365–366
file system connection, 368–369
FPE connection, 367–368
FTP connection, 366–367
transferring files, 369
Count( ) function, 107–108
counting returned items, 107–108
CreateUserWizard control, 278–280
CRUD (Create, Retrieve, Update, Delete),
47. See also FormView control;
GridView control
CSS (Cascading Style Sheets), 167–168,
196–200
currency, formatting, 109
cursor, changing control appearance. See
rollover effects
custom characters, 247–249. See also
reserved characters
CustomValidator control, 324–326

D

data
adding to a database. See inserting data
consuming. See consuming data
deleting
confirming, 126–127
DeleteCommand attribute, 83
Northwind database, 85, 124–126
SqlDataSource control, 83
handling. See SqlDataSource control
updating. See also editing; inserting
changing existing data, 139–140
custom images, 231–232
DataContext classes, 140–141
extension methods, 138
inserting data, 140–141
SqlDataSource control, 82
ToExclaim( ) method, 138–140
UpdateCommand attribute, 83
user interface to. See consuming data
data binding, 210–211
data source, specifying, 82
data types, checking, 321–322
databases. See also Northwind database; SQLExpress
adding data. See inserting data
connecting to. See also LinqDataSource control; SqlDataSource control
Access, 151–152
LINQ to SQL, 123
Northwind database, 79–80
SQLExpress, 77–80
drilling down to details. See master/detail pages
fetching data, 43–45. See also querying databases; SqlDataSource control
filtering records. See also parameters, passing
* (asterisk), wildcard character, 105
by character string, 104–106
comparison operators, 106
Contains() function, 105
by date and time, 107
EndsWith() function, 105–106
Like keyword, 104–105
by numbers, 106–107
StartsWith() function, 105–106
Where clause, 103–104
wildcard matching, 105–106
XML files, 119–120
selecting records, 83, 86–87, 102–103
DataContext classes, 140–141, 212–213
DataContext object, 100
data-driven check boxes, 68–69
data-driven Web pages, 43–45, 53–54
DataPager control, 221–222
date and time
date display, formatting, 52–53
date-picker, 249–251
daylight savings time server, 157–159
filtering by, 107
input validation, 58–59
validating, 321–322
Visual Basic format, 107
daylight savings time server, 157–159
debugging. See also exception handling; troubleshooting
Autos pane, 344
breaking on conditions, 342–343
breakpoints, 337–339
compile-time errors, 336–337
design-time errors, 335–336
editing code during execution, 343–344, 345
Immediate pane, 345
keystroke commands, 347
Locals pane, 344
logic errors, 337–340
regression errors, 338
run-time errors, 340–342
sample code, 333–335
squiggly underlines, 335
stepping through code, 339–340
test-driven development, 338
toolbar, 347–348
tooltips for displayed errors, 335
tracing, 345–346
variables, 344
VWD panes, 344–345
Watch pane, 344
declarative markup, 82–83
decorating classes, 299
dedicated style rules, 199–200
DefaultButton property, 206
DefaultFocus property, 206
DeleteCommand attribute, 83
deleting
data
confirming, 126–127
DeleteCommand attribute, 83
Northwind database, 85, 124–126
SqlDataSource control, 83
rows, 50–51
style rules, 174
deploying Web sites
Copy Web Site tool, 365–369
encrypting connection information, 377
hosting, 374
precompiling, 376–377
source code, protecting, 376–377
SQL databases, 372–373
SQL Publishing Wizard, 369–372
troubleshooting, 374–376
trust issues, 374
dereferencing objects, 134
Design view, 27, 30
designers
Collection editor, 65–66
definition, 48
generating templated controls, 210–211
design-time errors, 335–336
Destination Folder screen, 24–25
DetailsView control. See also displaying, data; master/detail pages
configuring, 97–98
consuming data, 84–86
performance, 86
DHTML (Dynamic HTML), 16–17. See also HTML
disambiguating classes, 295
displaying
custom images, 231–232
data. See also consuming data;
DetailsView control; ListView control
alternating visual appearance, 216
in chunks, 110–111, 221–222
DataPager control, 221–222
item delimiters, 220
ItemTemplate, 215–216
settings, 29
source code, 27
thumbnail images, 238
displaying data. See also consuming data;
DetailsView control; ListView control
alternating visual appearance, 216
in chunks, 110–111, 221–222
DataPager control, 221–222
hierarchical
GridView control, inner, 137
GridView control, outer, 135–136
grouping with a query, 133–135
labeling categories, 136
as a list of hyperlinks, 190–191
in a treeview, 184–186
Xml DataSource control, 184–186
item delimiters, 220
ItemTemplate, 215–216
Distinct keyword, 111–115
<div> tags, 198–199, 252–253
Document Outline pane, 199
documents, delivering over the Web
Excel, 269
PDFs, 266–268
Word, 268–269
dot (.)
class-based style rules, 200
object initialization, 116
drag and drop, data from a database, 43–45
drop-down lists
creating, 70
dynamic items, 70
filling, 89
LINQ to SQL, 128–130
populating the list, 70–71
static items, 70
DropDownList control, 69–71
duplicates, eliminating, 113
DVD Web project. See Web site project
Dynamic HTML (DHTML), 16–17. See also HTML

• E •
eating exceptions, 350
editing code during execution, 343–344, 345
editing data. See also updating
Collection editor, 65–66
content editors, 10–11
EditItemTemplate, 54, 216–217
FCKeditor.Net, 63
GridView control, 50–51
RTE (Rich Text Editor), 63
text editors, 63
EditItemTemplate, 54, 216–217
e-mail, exception handling, 353–355
embedding
controls in templates, 353–355
styles, 167
EmptyDataTemplate, 54, 219–220
EnablePagingCallbacks property, 86
encrypting connection information, 377
endpoints, 157, 160–162
EndsWith( ) function, 105–106
entities, 194
equal sign (=), object initialization, 116
error messages, 359–361. *See also* exception handling
error stack traces, 351
errors. *See* debugging; exceptions; troubleshooting
escaping reserved characters, 194
event handlers, 65
events, 65
evil nature of user input, 316
Excel files, serving on the Web, 269
exception handling. *See also* debugging; troubleshooting
The application...operation not allowed... message, 359
default handler duplication, 361
definition, 350
e-mail exceptions, 353–355
error messages, 359–361
error stack traces, 351
Expression...not queryable message, 361
Finally blocks, 358–359
global, 351–353
local, 355–358
missing assembly references, 360–361
null references, 360
NullReferenceException, 360
Object reference not set... message, 360–361
security violations, 359
SecurityException, 359
Try...Catch blocks, 355–358
Try...Catch...End Try blocks, 357
The type or namespace name...does not exist... message, 360–361
exceptions, 350
exclamation mark, red, 42
exclamation marks (!!), extension methods, 138
expanding/collapsing, tree nodes, 181
expensive content, caching, 201–202
Expression Blend, 10–11
Expression Web, 10
Expression...not queryable message, 361
extenders, AJAX Control Toolkit always on top, 251–254
AlwaysVisibleControlExtender, 251–254
AutoCompleteExtender, 241–244
auto-completion feature, 241–244
CalendarExtender, 249–244
date-picker, 249–251
date-picker, 249–251
floating text over pages, 251–254
listing, 240–241
MaskedEditExtender, 247–248
masking text input, 246–248
performance improvement, 60
refreshing Web pages, 60
text box prompts, 244–248
TextBoxWatermarkExtender, 245–246
watermarks, 244–246
Extensible Markup Language (XML). *See* XML (Extensible Markup Language)
Extensible Stylesheet Language (XSL), 148–151
extensions to Ajax. *See* AJAX Control Toolkit, extenders
ASP.NET, 260
methods, 138
external styles, 168–170

• F •

FCKeditor.Net, 63
fetching data, 43–45. *See also* querying databases; SqlDataSource control
Fiddler2, 384
fields, database, 40
File Transfer Protocol (FTP), 366–367
filenames, nonalphabetic characters, 38
files and folders, organizing, 32–33
filtering records. *See also* parameters, passing
* (asterisk), wildcard character, 105
by category, 131–132
by character string, 104–106
comparison operators, 106
Contains( ) function, 105
by date and time, 107
EndsWith( ) function, 105–106
Like keyword, 104–105
LinqDataSource control, 131–132
by numbers, 106–107
StartsWith( ) function, 105–106
Where clause, 103–104
Where parameter, 132
wildcard matching, 105–106
XML files, 119–120
Finally blocks, 358–359
FKs (foreign keys), 85
Flash, 264–265
Flasher control, 264–265
float: left style rule, 199–200
floating text, 251–254
focus, at startup, 206
folder names, nonalphabetic characters, 38
folders, securing, 286–289
FooterItemTemplate, 54
For Each loops, identifying selected check boxes, 69
formatting
   AutoFormat feature, 49
   controls, 49
currency, 109
date display, 52–53
   with starter designs, 49
forms
   authentication, configuring, 276–277
   source code, 73–74
   state, recording, 74
   survey, 65–67
   ViewState, 74
FormView control
   adding to a page, 53–54
data source, setting, 53–54
   rows, creating, 56–57
templates, 54–56
forums.asp.net, 381–382
FPE (FrontPage Server Extensions), 367–368
From...In clause, 102
FTP (File Transfer Protocol), 366–367
functions
   Average( ), 108–109
   Contains( ), 105
   Count( ), 107–108
   EndsWith( ), 105–106
   Split( ), 157
   StartsWith( ), 105–106
   Sum( ), 109

• G •
get accessors, 297–298
Google, troubleshooting aid, 380
graphics. See images
GridView control
color, 49
configuring, 97–98
date display, formatting, 52–53
deleting rows, 50, 51
displaying, 137
inner, 137
inserting into Web pages, 44
outer, 135–136
sorting columns, 50, 51
Group By keyword, 111–115
grouping
colors, 326–327
   date display, 52–53
   with starter designs, 49
forms
   authentication, configuring, 276–277
   source code, 73–74
   state, recording, 74
   survey, 65–67
   ViewState, 74
FormView control
   adding to a page, 53–54
data source, setting, 53–54
   rows, creating, 56–57
templates, 54–56
forums.asp.net, 381–382
FPE (FrontPage Server Extensions), 367–368
From...In clause, 102
FTP (File Transfer Protocol), 366–367
functions
   Average( ), 108–109
   Contains( ), 105
   Count( ), 107–108
   EndsWith( ), 105–106
   Split( ), 157
   StartsWith( ), 105–106
   Sum( ), 109
• H •
handicapped users. See accessibility
hardcoding style values, 166–167
HeaderItemTemplate, 54
help. See also online resources;
   troubleshooting
colors, 326–327
data source, setting, 53–54
displaying, 137
inner, 137
inserting into Web pages, 44
outer, 135–136
sorting columns, 50, 51
Group By keyword, 111–115
language grouping page, 111–113
with a query, 113–114, 133–135
rendering on a Web page, 114–115
users. See roles
Guthrie, Scott, 13, 383

• H •
handicapped users. See accessibility
hardcoding style values, 166–167
HeaderItemTemplate, 54
help. See also online resources;
   troubleshooting
colors, 326–327
data source, setting, 53–54
displaying, 137
inner, 137
inserting into Web pages, 44
outer, 135–136
sorting columns, 50, 51
Group By keyword, 111–115
language grouping page, 111–113
with a query, 113–114, 133–135
rendering on a Web page, 114–115
users. See roles
Guthrie, Scott, 13, 383
HTML (Hypertext Markup Language). See also DHTML standards, 194–196
in text boxes, 328–329
transforming XML to, 148–151
validating, 195–196
XML compliance, 194

• I •
ID (identity) columns, 39–41
ID values, editing, 50
IDE (integrated development environment). See VWD (Visual Web Developer); VWDE (Visual Web Developer Express)
IIS (Internet Information Services), 19
images
alternate text for, 203
custom, 228–232
rollover effects, 225–227
thumbnails, 234, 236–238
uploading, 232–238
Immediate pane, 345
inheritance, 300
inline styles, 166–167
in-memory cookies, 276
InsertCommand attribute, 83
inserting
controls into Web pages, 28
GridView control, 44
LinkButton control, 306
SqlDataSource control, 81–82
inserting data. See also updating data
InsertCommand attribute, 83
InsertItemTemplate, 54–56, 218–219
LINQ to SQL, 140–141
ListView control, 218–219
Query Designer, 42–43
SqlDataSource control, 83
InsertItemTemplate, 54–56, 218–219
installing VWD. See VWD (Visual Web Developer), installing; VWDE (Visual Web Developer Express)
instantiating objects, 295
integrated development environment (IDE). See VWD (Visual Web Developer); VWDE (Visual Web Developer Express)
Internet Explorer Developer Toolbar, 384
Internet Information Services (IIS), 19
italic text, 63. See also rich content
item delimiters, 220
ItemSeparatorTemplate, 220
ItemTemplate, 54–56, 215–216
iterating through, .NET collections, 67
iteration variables
declaring, 102
dereferencing objects, 134
out of scope, 104

• J •
JavaScript. See also AJAX
attack defense, disabling, 328–329
case sensitivity, 257
client-side code, 14–15
rollover effects, 225–227
in text boxes, 319
JavaScript Alert box, 328

• K •
keyboard shortcuts. See access keys
keystroke commands, debugging, 347
Knabben, Frederico Caldeira, 63

• L •
Label control, 136
labeling data, 136
layout. See Web pages; specific elements
License Terms screen, 23–24
Like keyword, 104–105
LIKE operator, 87
line breaks, LINQ, 101
LinkButton control, 306–308
LINQ (Language Integrated Query)
_ (underscore), line break character, 101
aggregating data, 107–111
anonymous types, 101
Average( ) function, 108–109
Contains( ) function, 105
Count( ) function, 107–108
DataContext object, 100
description, 18
login page, 280–281
login/logout link, 284
LoginStatus control, 284

**M**

MaskedEditExtender, 247–248
masking user input, 246–248, 322–323
master pages. See also templates
  breadcrumbs, 190–191
  creating, 174–175
  selecting, 175
master/detail pages
  detail data, fetching, 96
  DetailsView control, configuring, 97–98
  drilling down to details, 93
GridView control, configuring, 97–98
  master data, fetching, 94–95
  page layout, designing, 94–95
Max( ) method, 109–110
MediaPlayer control, 263–264
Membership
  Admin folder, 285–288
  administration area, 284–286
  administrator role, 276
  administrators, adding, 288–289
  anonymous visitors, 275
  categories of users, 275–276
ChangePassword control, 283–284
CreateUserWizard control, 278–280
data base, creating, 275–278
  folders, securing, 286–289
  forms authentication, configuring, 276–277
in-memory cookies, 276
Login control, 280–281
login page, 280–281
login/logout link, 284
LoginStatus control, 284
members, 275
membership list page, 285–286
  pages, securing, 289–290
password change page, 283–284
password recovery page, 281–282
preparing a site for, 274–275
registration page, 278–280
requirements, identifying, 275
  roles, 277–279
user groups. See roles
Menu control, 186–189
menus, 186–189
methods
  Bind( ), 210–211
  definition, 295
  extensions to, 138
  Max( ), 109–110
  Min( ), 109–110
  overloading, 299
  signatures, 299
ToExclaim( ), 138–140
Microsoft Live Search, troubleshooting aid, 380
Microsoft .NET 3.5 Framework, 12
Microsoft Office, 10
Microsoft Word 2007, 10
Min( ) method, 109–110
minimum/maximum values, 109–110
money. See currency
mouse, changing control appearance. See rollover effects
movies. See rich content
moving style rules, 173–174
msnews.microsoft.com, 382
multimedia. See rich content
  multimedia plug-in, 17–18
music. See rich content

**N**

namespaces, 71, 295
navigation
  breadcrumbs, 190–191
  menus, 186–189
TreeView control
  nodes, 179–182
  SiteMapDataSource control, 183
Web.sitemap files, 182–183
System.Data.DataSource control, 184–186
.NET collections, 67
“no data” warning, 219–220
nodes, treeview, 179–182
nonalphabetic characters, in file/folder names, 38
Northwind database. See also databases; SQLExpress
    adding to an application, 78–79
    connecting to, 79–80
    deletion constraint, 85, 124–126
downloading, 78
FKs (foreign keys), 85
null references, 360
NullReferenceException, 360
nulls, allowing in databases, 40
numbers, filtering by, 106–107
• O •
Object reference not set... message, 360
ObjectDataSource control, 309–310
objects
definition, 295
dererencing, 134
initializing, 116–117
instantiating, 295
mapping to database tables, 212–213
on Web pages. See controls
Office, 10
ohnosecond, 51
OleDbException... error, 151–152
online resources. See also help;
    troubleshooting
blogs, 382–383
forums.asp.net, 381–382
free tools, 384
Google, 380
Internet Explorer Developer Toolbar, 384
Microsoft Live Search, 380
msnews.microsoft.com, 382
online help, 380
peer-to-peer support, 381–382
search engines, 380
Web sites, 383
Web-based reference material, 380–381
Order By keyword, 111–115
out of scope, iteration variables, 104
overloading methods, 299
• P •
page directives, 73
page layout. See Web pages; specific elements
PagerTemplate, 54
parameters, passing to classes, 298
queries
    from a drop-down list, 88–89
    from Session variables, 90–92
    specifying a source for, 89–90
    on SQL queries, 92–93
    from TextBox controls, 86–87
passwords
    changing, 283–284
    recovering, 281–282
text boxes, 62
PDFs, 266–268
peer-to-peer support, 381–382
performance
    caching expensive content, 201–202
    DetailsView control, 86
    load time, reducing, 200–202
    refreshing Web pages, 60
    turn off ViewState, 200–201
placeholder controls, 214–215
plug-ins. See extensions
postbacks, 14–15
precompiling deployed code, 376–377
previewing Web pages
    data-driven Web pages, 45
    Design view, 27
    Document Outline pane, 199
    in a Web browser, 28–29
primary key, database, 40–41
ProfileCommon class, 305
profiles
    anonymity, 305
    enabling, 305
    overview, 291–292
properties
    AccessKey, 206
classes, 295
    DefaultButton, 206
    DefaultFocus, 206
properties continued
   EnablePagingCallbacks, 86
   TabIndex, 205–206
   VWD, 33–35
   Properties window, 33–35
   punctuation, in file/folder names, 38
   pushpin, 30–31

• Q •

   querying databases. See also LINQ
duplicates, eliminating, 88–89
   LIKE operator, 87
   parameters, passing
      from a drop-down list, 88–89
      from Session variables, 90–92
   specifying a source for, 89–90
   on SQL queries, 92–93
   from TextBox controls, 86–87
   querying XML files, 119–120
   quotes, around inline statements, 209

• R •

   radio buttons, 63–67
   RadioButton control, 63–64
   RadioButtonList control, 64–67
   raising exceptions, 350
   range variables. See iteration variables
   RangeValidator control, 317–318
   red exclamation mark, 42
   reference documents, 380–381
   Reflector, 384
   refreshing Web pages, performance, 60
   registration page, 278–280
   regression errors, 338
   regular columns, creating, 41
   regular expressions, 322–324
   RegularExpressionValidator control, 322–324
   Repeater control, 208–209
   RequiredFieldValidator control, 316–317
   reserved characters, 194. See also custom
      characters
   rethrowing exceptions, 350
   reviewing Web pages. See previewing Web
      pages

rich content, interactive media
   Flash, 264–265
   Flasher control, 264–265
   MediaPlayer control, 263–264
   Silverlight
      downloading, 256
      hosting, 260–262
      overview, 17–18
      project setup, 256–258
      static XAML content, 258–259
      Windows Media files, playing, 262–263
rich content, text, 63
roles, Membership
   applying, 286–290
   confirming, 289
   creating, 277–278
   enabling, 277–278
rollover effects
   <a> (anchor tag), 223–225
   definition, 223
   with HTML stylesheets, 223–225
   images, 225–227
   with JavaScript, 225–227
   text, 223–225
roundtrips, 200
rows
   counting, 107–108
   creating, 42–43, 56–57. See also
      FormView control
deleting, 50–51
RTE (Rich Text Editor), 63
run-time errors, 340–342

• S •

sandbox, 17
scripts, 14. See also JavaScript
search engines, troubleshooting aid, 380
security
   authentication, 273–274. See also
      Membership
   authorization, 273–274
   encrypting connection information, 377
   evil nature of user input, 316
   exceptions, 359
   HTML in user input, 328–329
   JavaScript attack defense, disabling, 328–329
passwords
  changing, 283–284
  recovering, 281–282
text boxes, 62
source code, protecting, 376–377
user input, as attack vector, 316, 319
SecurityException, 359
Select clause, 102–103
SelectCommand attribute, 83, 86–87
selecting records
  Select clause, 102–103
  SelectCommand control, 83, 86–87
serializing classes, 299
Session variables
  configuring SqlDataSource for, 91
  expiration, 91
  passing parameters to queries, 90–92
  setting, 91
set accessors, 297–298
settings, displaying, 29
Setup Complete screen, 25
setup information, sending to Microsoft, 23
shopcart.aspx page, 309
shopping cart code
class file, creating, 296–297
classes. See also CartItem class;
  ShoppingCart class
base, 300
Calculations, 311–312
definition, 295
disambiguating, 295
get accessors, creating 297–298
parameters, 298
ProfileCommon, 305
properties, 295
set accessors, creating, 297–298
data handling, 309–310
instantiating objects, 295
methods, 295, 299
namespaces, 295
ObjectDataSource control, 309–310
objects, 295
shopcart.aspx page, 309
values, 295
XML item description, 295–296
shopping cart items
adding
  Add to Cart interface, 293
code for, 300–302
  LinkButton control, 306–308
  steps, 314
creating, 298–299
finding, 303
list of, getting, 303–304
quantity, updating, 314
removing, 302–303
  updating number of, 304
shopping carts
calculations, 311–313
columns, inserting, 313
content management, 309–311
subtotal cost, calculating, 313
taxes, calculating, 313
total cost, calculating, 313
tracking status, 293
updating, 311, 314
viewing contents, 294, 311
ShoppingCart class, 299–304. See also shopping cart code
shortcut keys. See access keys
signatures, methods, 299
Silverlight
downloading, 256
hosting, 260–262
overview, 17–18
project setup, 256–258
static XAML content, 258–259
Windows Media files, playing, 262–263
Silverlight control, 260–262
single file model, 43–44
SiteMap. See Web.sitemap files
SiteMapDataSource control, 183
SiteMapPath control, 190–191
skins, 176. See also themes
Skip( ) operator, 110–111
Small Business Starter Kit
administrative functions, 284–286
CSS example, 197–198
downloading, 274
installing, 274–275
shopping cart example, 292
Smart Tags, 48
Solution Explorer, 32–33
  sorting
    columns, 50, 51
    data, LINQ, 111–115
sound. See rich content
source code
  displaying, 27
  protecting, 376–377
Source view, 27
spaces, in file/folder names, 38
special characters. See custom characters;
  reserved characters
Split( ) function, 157
Split view, 27
SQL databases, 372–373
SQL Publishing Wizard
  database script, creating, 370–371
  description, 369–370
  remote databases, creating from scripts,
    371–372
SQL Server, 19
SqlDataSource control
  command attributes, 83
  configuring, 82–83
  connection string, example, 83
  ConnectionString attribute, 82
  data source, specifying, 82
  declarative markup, 82–83
  DeleteCommand attribute, 83
  deleting data, 83
  displaying data. See consuming data;
    DetailsView control
  fetching data, 82
  InsertCommand attribute, 83
  inserting data, 83
  inserting into Web pages, 81–82
  parameters, defining, 83
  parameters, passing to queries
    from a drop-down list, 88–89
    from Session variables, 90–92
    specifying a source for, 89–90
    on SQL queries, 92–93
  from TextBox controls, 86–87
  read-only configuration, 95
  SelectCommand attribute, 83, 86–87
  selecting data, 83, 86–87
  UpdateCommand attribute, 83
  updating data, 82–83
SQLExpress database. See also LINQ to SQL
  adding to a project, 38–39
  connecting to, 77–80
  data
    drag and drop to a page, 43–45
    fetching, 43–45
    inserting, 42–43
  primary key, setting, 40–41
  sample. See Northwind database
tables. See also columns; rows
  allow nulls, 40
  creating, 39
  data, inserting, 42–43
  fields, 40
  ID numbers, automatic, 40
  red exclamation automatic mark, 42
  verifying it is running, 77–78
  squiggly underlines, 335
stack, 12
standards, 194–196
starter designs, 49. See also templates
starter kits, troubleshooting aid, 382
StartsWith( ) function, 105–106
startup focus, 206
state of forms, recording, 74
stepping through code, 339–340
storing styles
  embedding, 167
  in external CSS style sheets, 167–168
  inline, 166–167
  in the <style> tag, 167
  applying, 166–167
  AutoFormat, 166
  built-in, 166
  class-based, 200
  dedicated, 199–200
  deleting, 174
  float: left, 199–200
  modifying, 174
  moving, 173–174
  <style> tag, storing styles in, 167
styles. See also master pages; templates;
  XSL
  applying, 166–167
  AutoFormat, 166
  built-in, 166
  AutoFormat, 166
  AutoFormat, 166
external, 168–170
hardcoding values, 166–167
storing, 167–168
on TextBox controls, 170–172
stylesheetTheme attribute, 178
subroutines. See methods
Sum( ) function, 109
summing returned values, 109
survey form, 65–67
symbols for reserved characters. See entities

• T •
tab order, setting, 205
TabIndex property, 205–206
table output, accessibility guidelines, 203
tables. See also columns; databases; rows adding data. See inserting data
allow nulls, 40
creating, 39
fields, 40
ID numbers, automatic, 40
red exclamation mark, 42
Take( ) operator, 110–111
technologies
See also AJAX
See also HTML
See also JavaScript
See also LINQ
See also Silverlight
See also Web services
See also XML
ADO.NET, 19
ASP.NET 3.5, 12–13
ASP.NET 3.5 Extensions, 13–14
ASP.NET Futures, 13
client-side code, 14–15
DHTML (dynamic HTML), 16–17
IIS (Internet Information Services), 19
Microsoft .NET 3.5 Framework, 12
multimedia plug-in, 17–18
SQL Server, 19
Web server, 19
templated controls
Bind( ) method, 210–211
data binding, 210–211
definition, 207
generating with designers, 210–211
list of, 207
ListView
alternating visual appearance, 216
AlternatingItemTemplate, 216
chunking data, 221–222
DataContext classes, generating, 212–213
DataPager, 221–222
displaying data, 215–216
editing records, 216–217
EditItemTemplate, 216–217
EmptyDataTemplate, 219–220
inserting records, 218–219
InsertItemTemplate, 218–219
item delimiters, 220
ItemSeparatorTemplate, 220
ItemTemplate, 215–216
LayoutTemplate, 214–215
LinqDataSource, configuring, 213
list format, 220–221
mapping objects to database tables, 212–213
“no data” warning, 219–220
placeholder controls, 214–215
setting up, 214
Repeater, 208–209
two-way binding, 210
templates
See also master pages
See also starter designs
See also styles
See also templated controls
AlternatingItemTemplate, 216
definition, 33
EditItemTemplate, 54, 216–217
embedding controls in, 207
EmptyDataTemplate, 54, 219–220
FooterItemTemplate, 54
FormView control, 54–56
HeaderItemTemplate, 54
InsertItemTemplate, 54–56, 218–219
ItemSeparatorTemplate, 220
ItemTemplate, 54–56, 215–216
LayoutTemplate, 214–215
PagerTemplate, 54
test-driven development, 338
testing, editing data, 50–51
text
bold, 63
floating, 251–254
italic, 63. See also rich content
text effects, 223–225
text boxes
bold text, 63
creating, 62
custom characters, 247–249
guiding input, 246–248
HTML markup, 328–329
italic text, 63
masking input, 246–248
multi-line input, 62, 323–324
passing parameters to queries, 86–87
passwords, 62
prompting for input, 244–248
restricting input, 246–248
rich text, 63
size, restricting, 62
text length, validating, 323–324
watermarks, 244–246
text editors, 63
TextBox control, 61–63, 170–172
TextBoxWatermarkExtender, 245–246
theme attribute, 178
themes, 176–178. See also skins
throwing exceptions, 350
thumbnails, 234, 236–238
time. See date and time
titles, Web pages, 59
ToExclaim( ) method, 138–140
Toolbox
AJAX Control Toolkit, 31–32
Auto Hide, 30–31
controls, adding, 31–32
making visible, 30–31
pushpin, 30–31
tools for
content creation, 10–11. See also
tools for
troubleshooting, 384
 tooltips, for displayed errors, 335
tracing errors, 345–346
trapping exceptions, 350

TreeView control
nodes, 179–182
SiteMapDataSource control, 183
Web.sitemap files, 182–183
XMLDataSource control, 184–186
troubleshooting. See also debugging;
exception handling; help
deployment problems, 374–376
resources for, 380–384
when you’re stuck, 379–380
ture/false logic, 235
Try...Catch blocks, 355–358
Try...Catch...End Try blocks, 357
two-file model, 44
two-way binding, 210
The type or namespace
name...does not exist...
message, 360–361

• U •
underscore (_), line break character, 101
undoing, deleted rows, 51
UpdateCommand attribute, 83
updating data. See also editing; inserting
changing existing data, 139–140
custom images, 231–232
DataContext classes, 140–141
extension methods, 138
inserting data, 140–141
SqlDataSource control, 82
ToExclaim( ) method, 138–140
UpdateCommand attribute, 83
uploading images. See images, uploading
URLs, keeping current, 156
usability
access keys, 205–206
AccessKey property, 206
default buttons, 206
DefaultButton property, 206
DefaultFocus property, 206
drop-down lists, 128–130
for handicapped users. See accessibility
LINQ to SQL, 127–130
LinqDataSource control, 127–130
startup focus, 206
tab order, 205
TabIndex property, 206
using names for numbers, 127–128
user groups. See roles
user input
check boxes, 68–69
drop-down lists
creating, 70
dynamic items, 70
filling, 89
LINQ to SQL, 128–130
populating the list, 70–71
static items, 70
radio buttons, 63–64
text boxes
bold text, 63
creating, 62
custom characters, 247–249
guiding input, 246–248
HTML markup, 328–329
italic text, 63
masking input, 246–248
multi-line input, 62, 323–324
passing parameters to queries, 86–87
passwords, 62
prompting for input, 244–248
restricting input, 246–248, 323–324
rich text, 63
size, restricting, 62
text length, validating, 323–324
watermarks, 244–246
validating
alphabetical data, 321–322
as attack vectors, 316, 319
with code, 324–326
comparing values, 319–321
data type checking, 321–322
date and time, 58–59, 321–322
evil nature of, 316
forcing an entry, 316–317
by groups, 326–327
HTML markup, 328–329
JavaScript, 319
JavaScript Alert box, 328
masking, 246–248, 322–323
numerical data, 321–322
prompting for input, 244–248
range checking, 317–318
with regular expressions, 322–324
summarizing invalid fields, 327–328
text length, 323–324
user interface. See also controls; specific elements
creating, 123–124
ease of use. See usability
generating from a database, 44–45. See also consuming data; DetailsView control
LINQ to SQL
creating, 123–124
drop-down lists, 128–130
names for numbers, 127–128
usability, 127–130

• V •
validating
accessibility, 204
HTML (Hypertext Markup Language), 195–196
user input
alphabetical data, 321–322
as attack vectors, 316, 319
with code, 324–326
comparing values, 319–321
data type checking, 321–322
date and time, 58–59, 321–322
evil nature of, 316
forcing an entry, 316–317
by groups, 326–327
HTML markup, 328–329
JavaScript, 319
JavaScript Alert box, 328
masking, 246–248, 322–323
numerical data, 321–322
prompting for input, 244–248
range checking, 317–318
with regular expressions, 322–324
summarizing invalid fields, 327–328
text length, 323–324
XHTML, 196
ValidationSummary control, 327–328
values, definition, 295
variables
debugging, 344
declaring, 297
video. See rich content
videos, troubleshooting aid, 383
ViewState, 74, 200–201
VWD (Visual Web Developer)
description, 11
Design view, 27, 30
hidden members, unhiding, 29–30
installing
Destination Folder screen, 24–25
downloading, 22–23
installation location, specifying, 24–25
Installation Options screen, 24
License Terms screen, 23–24
optional products, selecting, 24
rebooting, 25
Setup Complete screen, 25
setup information, sending to Microsoft, 23
versus VWD (Visual Web Developer), 21
Welcome to Setup screen, 23
organizing files and folders, 32–33
properties, 33–35
Properties window, 33–35
settings, displaying, 29
Solution Explorer, 32–33
Source view, 27
Split view, 27
starter designs, 49. See also templates
starting, 26
templates, 33
Toolbox, 30–32
VWDE (Visual Web Developer Express)
description, 11
Design view, 27, 30
hidden members, unhiding, 29–30
installing
Destination Folder screen, 24–25
downloading, 22–23
installation location, specifying, 24–25
Installation Options screen, 24
License Terms screen, 23–24
optional products, selecting, 24
rebooting, 25
Setup Complete screen, 25
W3C (World Wide Web Consortium)
accessibility guidelines, 203
XHTML validation, 196
Watch pane, 344
watermarks, 244–246
WCF (Windows Communication Foundation)
binding, 157
connecting to, 160–162
consumer for, creating, 159–160
contracts, 157
daylight savings time server, 157–159
endpoints, 157, 160–162
Web Development Helper, 384
Web pages. See also specific elements
buttons. See controls
consistent appearance. See also styles
master pages, 174–175
skins, 176
templates, 33, 54–56
themes, 176–178
creating, 26–29
data driven. See data-driven Web pages
objects on. See controls
previewing
data-driven Web pages, 45
Design view, 27
Document Outline pane, 199
in a Web browser, 28–29
refreshing, performance, 60
securing, 289–290. See also Membership
source code, displaying, 27
titles, changing, 59
Web references, 155–156
Web server, 19, 46
Web Service Description Language (WSDL), 160–162
Web services
  connecting to, 155–156
  consumer for, creating, 156–157
  creating, 152–154
  description, 14
  on networks. See WCF
Web site project
  creating, 37–38
  database, adding. See SQLExpress
database
data-driven Web pages, 43–45
Web sites
  creating, 26–27
  deploying. See deploying Web sites
  for developers. See online resources
troubleshooting aid, 383
Web-based reference material, 380–381
web.config file, connectionStrings
  section, 83
WebHandler, 236–238
Web.sitemap files, 182–183, 188–189
Welcome to Setup screen, 23
well-formed XML, 194
Where clause, 103–104
Where parameter, 132
wildcard matching, 105–106
Windows Communication Foundation
  (WCF). See WCF (Windows
Communication Foundation)
Windows Media files, playing, 262–263
Word documents, serving on the Web,
  268–269
World Wide Web Consortium (W3C)
  accessibility guidelines, 203
  XHTML validation, 196
  wrapping exceptions, 350
  WSDL (Web Service Description Language),
    160–162

X

XHTML standards, 194–196
XML (Extensible Markup Language). See
  also AJAX; LINQ to XML
case consistency, 194
closing tags, 194
description, 17
entities for reserved characters, 194
overview, 17
quote attributes, 194
self-closing tags, 194
standards, 194
style sheets. See XSL
transforming to HTML, 148–151
well-formed rules, 194
XML control, 150–151
XmlDataSource control, 144, 184–186
XSL (Extensible Stylesheet Language),
  148–151
XSP.NET, 13

Z

zero-based numbering, 67
SPORTS, FITNESS, PARENTING, RELIGION & SPIRITUALITY

Also available:
- Catholicism For Dummies 0-7645-5391-7
- Exercise Balls For Dummies 0-7645-5623-1
- Fitness For Dummies 0-7645-7851-0
- Football For Dummies 0-7645-3936-1
- Judaism For Dummies 0-7645-5299-6
- Potty Training For Dummies 0-7645-5417-4
- Buddhism For Dummies 0-7645-5359-3

TRAVEL

Also available:
- Alaska For Dummies 0-7645-7746-8
- Cruise Vacations For Dummies 0-7645-6941-4
- England For Dummies 0-7645-4276-1
- Europe For Dummies 0-7645-7529-5
- Germany For Dummies 0-7645-7823-5
- Hawaii For Dummies 0-7645-7402-7
- Italy For Dummies 0-7645-7386-1
- Las Vegas For Dummies 0-7645-7382-9
- London For Dummies 0-7645-4277-X
- Paris For Dummies 0-7645-7630-5
- RV Vacations For Dummies 0-7645-4442-X
- Walt Disney World & Orlando For Dummies 0-7645-9660-8

GRAPHICS, DESIGN & WEB DEVELOPMENT

Also available:
- 3D Game Animation For Dummies 0-7645-8789-7
- AutoCAD 2006 For Dummies 0-7645-8925-3
- Building a Web Site For Dummies 0-7645-7144-3
- Creating Web Pages For Dummies 0-470-0830-2
- Creating Web Pages All-in-One Desk Reference For Dummies 0-7645-4345-8
- Dreamweaver 8 For Dummies 0-7645-9649-7
- InDesign CS2 For Dummies 0-7645-9572-5
- Macromedia Flash 8 For Dummies 0-7645-9691-8
- Photoshop CS2 and Digital Photography For Dummies 0-7645-9580-6
- Photoshop Elements 4 For Dummies 0-471-77483-9
- Syndicating Web Sites with RSS Feeds For Dummies 0-7645-8848-6
- Yahoo! SiteBuilder For Dummies 0-7645-9800-7

NETWORKING, SECURITY, PROGRAMMING & DATABASES

Also available:
- Access 2007 For Dummies 0-470-04612-0
- ASP.NET 2 For Dummies 0-7645-7007-X
- C# 2005 For Dummies 0-7645-9704-3
- Hacking For Dummies 0-470-05235-X
- Hacking Wireless Networks For Dummies 0-7645-9730-2
- Java For Dummies 0-470-08716-1
- Microsoft SQL Server 2005 For Dummies 0-7645-7755-7
- Networking All-in-One Desk Reference For Dummies 0-7645-9939-9
- Preventing Identity Theft For Dummies 0-7645-7336-5
- Telecom For Dummies 0-471-77085-X
- Visual Studio 2005 All-in-One Desk Reference For Dummies 0-7645-9775-2
- XML For Dummies 0-7645-8845-1

Also available:
- Adobe Creative Suite 2 For Dummies 0-7645-8815-X
- Dreamweaver 8 For Dummies 0-7645-9649-7
- Photoshop CS2 For Dummies 0-7645-9571-7
- InDesign CS2 For Dummies 0-7645-9572-5
- Macromedia Flash 8 For Dummies 0-7645-9691-8
- Photoshop CS2 and Digital Photography For Dummies 0-7645-9580-6
- Photoshop Elements 4 For Dummies 0-471-77483-9
- Syndicating Web Sites with RSS Feeds For Dummies 0-7645-8848-6
- Yahoo! SiteBuilder For Dummies 0-7645-9800-7

Also available:
- Access 2007 For Dummies 0-470-04612-0
- ASP.NET 2 For Dummies 0-7645-7007-X
- C# 2005 For Dummies 0-7645-9704-3
- Hacking For Dummies 0-470-05235-X
- Hacking Wireless Networks For Dummies 0-7645-9730-2
- Java For Dummies 0-470-08716-1
- Microsoft SQL Server 2005 For Dummies 0-7645-7755-7
- Networking All-in-One Desk Reference For Dummies 0-7645-9939-9
- Preventing Identity Theft For Dummies 0-7645-7336-5
- Telecom For Dummies 0-471-77085-X
- Visual Studio 2005 All-in-One Desk Reference For Dummies 0-7645-9775-2
- XML For Dummies 0-7645-8845-1
Do More with Dummies

Instructional DVDs • Music Compilations
Games & Novelties • Culinary Kits
Crafts & Sewing Patterns
Home Improvement/ DIY Kits • and more!

Check out the Dummies Specialty Shop at www.dummies.com for more information!