

Start creating Web pages now
with Microsoft's® Visual Web Developer

Visual Web Developer 2005 Express Edition

FOR
DUMMIES®

**A Reference
for the
Rest of Us!**

FREE eTips at dummies.com

Get up and running
quickly to deploy
Web applications



Alan Simpson

TEAM LinG

Visual Web Developer™
2005 Express Edition

FOR
DUMMIES®

Visual Web Developer™
2005 Express Edition
FOR
DUMMIES®

by Alan Simpson



WILEY

Wiley Publishing, Inc.

Visual Web Developer™ 2005 Express Edition For Dummies®

Published by
Wiley Publishing, Inc.
111 River Street
Hoboken, NJ 07030-5774

www.wiley.com

Copyright © 2006 by Wiley Publishing, Inc., Indianapolis, Indiana

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4355, or online at <http://www.wiley.com/go/permissions>.

Trademarks: Wiley, the Wiley Publishing logo, For Dummies, the Dummies Man logo, A Reference for the Rest of Us!, The Dummies Way, Dummies Daily, The Fun and Easy Way, Dummies.com, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates in the United States and other countries, and may not be used without written permission. Visual Web Developer is a trademark of Microsoft Corporation in the United States and/or other countries. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

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

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

For technical support, please visit www.wiley.com/techsupport.

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

Library of Congress Control Number: 2005927626

ISBN-13: 978-0-7645-8360-5

ISBN-10: 0-7645-8360-3

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

1B/RQ/RR/QV/IN



About the Author

Alan Simpson is the author of over 90 computer books on databases, Windows, Web-site design and development, programming, and networking. His books are published throughout the world in over a dozen languages. Alan has served as a consultant on high-technology projects for the United States Navy and Air Force.

Dedication

To Susan, Ashley, and Alec.

Author's Acknowledgments

Writing a book is always a team effort, and this book is no exception. I'd like to take this opportunity to thank all the folks who made this book possible, and contributed to its completion. At Wiley Publishing, many thanks to Katie Feltman for providing the opportunity (and reminders to get on schedule). Thanks to Christopher Morris, Barry Childs-Helton, and Dan DiNicolo for their superior editing.

Thanks to David Fugate at Waterside Productions, my literary agency, for getting the ball rolling and ironing out the details.

And most of all, thanks to my family for putting up with yet another long Daddy project.

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: Christopher Morris
Acquisitions Editor: Katie Feltman
Senior Copy Editor: Barry Childs-Helton
Technical Editor: Dan DiNicolo
Editorial Manager: Kevin Kirschner
Media Development Manager:
Laura VanWinkle
Media Development Supervisor:
Richard Graves
Editorial Assistant: Amanda Foxworth
Cartoons: Rich Tennant
(www.the5thwave.com)

Composition Services

Project Coordinator: Adrienne Martinez
Layout and Graphics: Carl Byers, Andrea Dahl,
Lauren Goddard, Barbara Moore,
Barry Offringa
Proofreaders: Laura Albert, Dwight Ramsey,
TECHBOOKS Production Services
Indexer: Sherry Massey

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

<i>Introduction</i>	1
<i>Part I: Planning a Web Site</i>	7
Chapter 1: Getting Started	9
Chapter 2: Creating a Web Site	21
Chapter 3: Configuring a Membership Site	39
Chapter 4: Creating Master Pages	53
<i>Part II: Building Your Web Site</i>	73
Chapter 5: Creating Web Pages	75
Chapter 6: Designing with Styles	97
Chapter 7: Working with ASP.NET Controls	123
Chapter 8: Easy Site Navigation	153
<i>Part III: Personalization and Databases</i>	169
Chapter 9: Using Personalization	171
Chapter 10: Using Themes	199
Chapter 11: SQL Server Crash Course	221
Chapter 12: Using Data in Web Pages	261
<i>Part IV: The Part of Tens</i>	319
Chapter 13: Ten Terms to Make You Look Smart	321
Chapter 14: Ten Alternatives to Being Helpless	327
<i>Appendix: Publishing Your Site</i>	331
<i>What's on the CD-ROM?</i>	337
<i>Index</i>	341
<i>End-User License Agreement</i>	Back of Book

Table of Contents

<i>Introduction</i>	1
About This Book	2
Foolish Assumptions	2
Conventions Used in This Book	3
What You're Not to Read	3
How This Book Is Organized	4
Part I: Planning a Web Site	4
Part II: Building Your Web Site	4
Part III: Personalization and Databases	4
Part IV: The Part of Tens	4
Icons Used in This Book	5
Where to Go from Here	5
<i>Part I: Planning a Web Site</i>	7
Chapter 1: Getting Started	9
Who VWD Is For	9
Installing Visual Web Developer	10
Getting Around in VWD	10
Using panes	11
Getting panes back to normal	12
Don't forget the View menu	13
About the Start Page	14
Using VWD Help	14
Closing Help pages and panes	16
Online resources	16
Being Compatible with Web Browsers	17
Publishing Your Web Site	19
Chapter 2: Creating a Web Site	21
Creating a Web Site	21
Closing and Opening Pages	23
Creating and Using Folders	24
Copying files to folders	25
Renaming and deleting folders	26

Editing Pages	26
Adding text to a page	27
Selecting and formatting text	27
Undoing changes	28
Adding pictures	28
Changing properties	29
Switching views	30
Editing in Source view	31
Saving your work	32
Dealing with code-behind files	33
Titling Pages	34
Viewing Pages in a Web Browser	35
Opening and Closing Web Sites	37
Chapter 3: Configuring a Membership Site	39
Creating a Folder for Members-Only Content	39
Using the Web Site Administration Tool	40
Choosing an authentication type	42
Creating Roles to Categorize People	43
Creating Access Rules	45
Managing access rules	46
Creating a User Account	48
Managing user accounts	50
Closing the Web Site Administration tool	51
What the Web Site Administration Tool Did	51
Chapter 4: Creating Master Pages	53
Creating a Folder for Master Pages	54
Creating a Master Page	54
Designing your page layout	55
Styling Master Page panes	57
Styling the left pane	60
Styling the ContentPlaceHolder pane	61
Using a Master Page	63
Editing a Master Page	66
Adding a Master Page to Existing Pages	69
Part II: Building Your Web Site	73
Chapter 5: Creating Web Pages	75
Creating a New Blank Page	75
Creating HTML Tables	77
Adding a table to a page	77
Typing in table cells	78

Working with HTML Tables	78
Selecting rows and columns	79
Selecting cells	80
Merging cells	80
Styling cells	81
Adding controls to table cells	84
Adding Hyperlinks to Pages	84
Quick links to pages in your site	85
Creating bookmarks	86
Linking to bookmarks	86
Adding and Styling Pictures	87
Sizing a picture	88
Styling pictures	88
Adding Lines	92
Editing in Source View	92
Selecting in Source view	93
Typing tags and attributes	93
Debugging HTML	95

Chapter 6: Designing with Styles97

Understanding CSS	97
Creating a CSS Style Sheet	100
Creating Style Rules	101
Creating CSS element styles	101
Creating CSS class selectors	102
Defining Rules with Style Builder	104
Styling fonts	105
Styling the background	107
Styling text alignment and spacing	108
Styling position	110
Styling layout	112
Styling boxes and borders	113
Saving Style Builder choices	114
Saving a CSS style sheet	115
Linking to a Style Sheet	115
Using Styles in a Page	116
Applying CSS element selectors	116
Applying CSS class selectors	117
Applying element class selectors	119
Using DIV styles	120
The CSS 2.1 Specification	122

Chapter 7: Working with ASP.NET Controls123

What Is ASP.NET?	123
Adding a Server Control to a Page	125
Tweaking server controls in Design view	126
Using the Common Tasks menu	127

ASP.NET Login Controls	130
Allowing Users to Create an Account	131
Assigning new users to a role	133
Testing the control	134
Creating a Login Page	135
Providing a Login Link	136
The LoginStatus control	137
The LoginName control	138
The LoginView control	138
Letting Users Manage Passwords	141
Using the PasswordRecovery control	141
The ChangePassword control	145
Testing Membership	146
Server Controls in Source View	148
Relaxing Password Constraints	149

Chapter 8: Easy Site Navigation 153

Getting Organized	153
Using Site-Navigation Controls	154
Using the TreeView and Menu Controls	155
Creating a Site Map	158
Customizing navigation for roles	161
Binding a control to Web.sitemap	163
Adding an Eyebrow Menu	164
Creating Web User Controls	165
Creating a Web User Control	166
Using a Web User Control	167

Part III: Personalization and Databases 169

Chapter 9: Using Personalization 171

Creating a User Profile	171
Setting up user profiles	173
Letting Users Enter Properties	176
Adding a button	178
Writing some code	179
Tying code to an event	180
Determining where to put the profile information	183
Letting users edit their profiles	184
Using profile properties with Visual Basic	187
Using Validation Controls	188
RequiredFieldValidator	189
RangeValidator	190

RegularExpressionValidator190
 CompareValidator191
 CustomValidator192
 ValidationSummary192
 Using the Forms Designer193
 Stacking absolutely-positioned objects194
 Aligning absolutely-positioned objects195
 Sizing objects equally196
 Spacing absolutely-positioned objects197

Chapter 10: Using Themes 199

Creating Themes199
 Creating Theme Folders200
 What's in a Theme?201
 Using Pictures in Themes201
 Creating a Theme Style Sheet202
 Creating Skins204
 Creating a skin file204
 Default vs. named skins207
 Using Themes in Pages209
 Letting Members Choose a Theme210
 Creating a page for viewing themes211
 Creating a control for choosing a theme212
 Storing the preferred theme213
 Applying a theme214
 A theme tester page216
 Applying Themes to Master Pages217
 Other Ways to Apply Themes218
 Defining a Site-Wide Default Theme219

Chapter 11: SQL Server Crash Course 221

Crash Course in Database Design222
 Tables, rows, and columns222
 One-to-many, many-to-many223
 SQL Server Tables227
 Assigning GUIDs automatically233
 Creating Your Own Tables236
 Defining a primary key237
 Creating text fields238
 Adding a money field240
 Saving the new table240
 Creating the Transactions table241
 A Primary Key for Transactions243
 Populating Tables244

Linking Tables	247
Creating a view	248
A more detailed view	251
Creating a Table of Pictures	254
Creating a Table of HyperLinks	257
Chapter 12: Using Data in Web Pages	261
Binding Data to Controls	262
Using the Data Configuration Wizard	262
Data controls in Design view	273
Formatting Dates and Numbers	274
Some Security Considerations	275
Using the GridView Control	276
An instant GridView control	276
Formatting the GridView control	278
Binding to DropDownList Controls	280
Using a DropDownList to filter records	282
Viewing and editing user properties	284
Using the DetailsView Control	287
Binding a DetailsView control	287
Formatting a DetailsView control	289
Creating Master-Details Forms	291
Master-Details DropDownList control	292
Master-Details GridView control	293
The Master-Details DetailsView control	294
General GridView and DetailsView considerations	295
Using the DataList Control	296
Formatting a DataList control	298
Formatting dates and numbers in a DataList	300
Showing a DataList in columns	301
Using DataList to show pictures	302
Using a DataList to show HyperLinks	309
The FormView Control	312
Showing subtotals	314
 Part IV: The Part of Tens	 319
Chapter 13: Ten Terms to Make You Look Smart	321
Web Application	321
Developer	321
Data-Driven	322
ASP.NET 2.0	322
Visual Studio	322
IDE	322

Control	323
Code	323
Programmatic	324
Database	325
Chapter 14: Ten Alternatives to Being Helpless	327
Microsoft Developer Network (MSDN)	327
HTML Home Page	327
Cascading Style Sheets Home Page	328
XML Home Page	328
ASP.NET	328
ASP.NET Starter Kits	328
ASP.NET Forums	329
SQL Server Developer Center	329
dotnetjunkies	329
Microsoft Technical Communities	329
 <i>Appendix: Publishing Your Site</i>	 <i>331</i>
Choosing a Hosting Provider	331
Preparing Your Site for Uploading	332
Copying the Site	334
 <i>What's on the CD-ROM?</i>	 <i>337</i>
Visual Basic 2005 Express	337
Visual Web Developer 2005 Express	338
 <i>Index</i>	 <i>341</i>
 <i>End-User License Agreement</i>	 <i>Back of Book</i>



Introduction

Welcome to *Visual Web Developer 2005 Express Edition For Dummies*. Visual Web Developer is a tool for developing dynamic, data-driven Web sites. The *dynamic* part refers to the fact that each page your site sends out can be tailored — on the spot and even on the fly — for whatever person happens to be viewing the page at that moment. The *data-driven* part stems from the fact that the information needed to create pages dynamically is stored in a database.

Historically, creating a data-driven Web site was an enormous task, requiring countless hours of tedious programming and debugging. Visual Web Developer (VWD) changes all that — allowing you to create dynamic Web sites in a quicker, easier, and more intuitive visual mode where simple drag-and-drop replaces hours of typing code.

That's not to say that Visual Web Developer is *so* easy that you can just “think” a Web site into existence. There's still plenty of knowledge and skill required to develop a Web site. You just don't need *as much* knowledge and skill as was required in the pre-VWD olden days.

If you've spent much time online trying to figure out how to work Visual Web Developer, you've probably been overwhelmed by countless buzzwords and confusing computer code that looks like something written by aliens from another planet. Much of what you've seen probably comes from people comparing the VWD way of doing things to the old way of doing things (and that can get obscure in a hurry).

This book takes a different approach: I don't talk about the old way of doing things at all. There are two reasons for that. The first is, if you don't know about the old way of doing things, then the comparisons don't help one bit. And if you *do* know the old way of doing things, then you can see for yourself how the new way is different without my telling you.

At the risk of sounding smarmy, I might even go so far as to say that the old way of doing this is irrelevant now. By the time you've finished with this book, you'll see what I mean. And you'll be able to create powerful data-driven Web sites — perhaps without typing any code at all.

About This Book

The main goal of this book is simple: To cover everything you really *need* to know about Visual Web Developer to create data-driven Web sites. And I do mean “need.” You won’t catch me wandering off into irrelevant product comparisons or advanced topics that few people need.

Another key ingredient of this book is its coverage of things that most other resources assume you already know — in fact, it’s okay if you don’t already know them. Everybody has to start somewhere, and Web-site development is tricky enough without having to fight a feeling of being left out. You won’t get, “Sorry, you didn’t learn our secret language umpteen years ago when we did, so you can’t play.” Here, just about everyone gets to play.

It’s important, especially for newbies, to understand that there’s a big difference between “everything you need to know (just to get in the game),” and everything there *is* to know. This book makes no attempt to cover everything there is to know about Visual Web Developer (as you’ll notice right away because one person can actually carry the book). The reason is simple: Ten books the size of this one couldn’t cover everything there is to know about Visual Web Developer. So you may need to rely on other resources from time to time. And that’s okay too.

Finally, though I’d like to be able to write this book in such a way that even a fresh-minted PC newbie could follow along, such a goal is unrealistic. Covering everything from your first mouse click to publishing your dynamic data-driven Web site would take too much space — so I have to make some assumptions about who is going to read this book. Which brings us to . . .

Foolish Assumptions

Creating dynamic data-driven Web sites, even with Visual Web Developer, is not a topic for absolute computer beginners. If you just got your first PC a few weeks ago, and so far have mastered only the art of e-mail, you may need to spend quite a bit more time learning Windows basics before you can tackle some of the terms used in this book without getting a headache.

It would be best if you already had some experience creating Web pages on your own. There isn’t really room in this book to discuss things like HTML and CSS in depth. So if those two acronyms are completely foreign to you, then you might want to study up on them before you start using this book.

On the bright side, you don’t really need to know anything, well, *technical* about ASP.NET or C# or SQL Server to use this book. You’ll use those technologies to create your site, sure enough, but I’ll give you a practical briefing

in how to boss them around. You don't have to be a mechanic to drive; likewise, you don't have to be a programmer or database developer already before you use this book.

Conventions Used in This Book

As you browse through this book you might notice some unfamiliar symbols or odd-looking text in gray boxes. Don't worry about 'em. The ⇔ symbol you'll see in the text just separates individual menu options (commands) that you choose in sequence. For example, rather than saying "Choose New from the File menu" or "Click File in the menu bar, then click New in the drop-down menu," I just say something like this:

Choose File ⇔ New from the menu bar.

Creating VWD Web sites doesn't take much programming. What little code is actually required to perform some task is shown in a `monospace` font, like this:

```
if (!Page.IsPostBack) {
    txtFirstName.Text = Profile.FirstName;
    txtLastName.Text = Profile.LastName;
    txtAddress1.Text = Profile.Address1;
    txtAddress2.Text = Profile.Address2;
    txtCity.Text = Profile.City;
    txtStateProv.Text = Profile.StateProvince;
    txtZipPostalCode.Text = Profile.ZIPPostalCode;
    txtPrefTheme.Text = Profile.PreferredTheme;
}
```

When there are a few little chunks of code to show in text, like this — `Profile.FirstName` — I show them that way so you can see what is and what isn't code.

What You're Not to Read

Reading computer books is not most people's idea of fun. (though it can be a great cure for insomnia, should the need ever arise). Any text that doesn't clearly fit into the need-to-know category of using VWD will be marked with Technical Stuff icons (more about those in a minute) or placed in gray sidebars. If you're in a hurry, or just feel overwhelmed by the need-to-know stuff, you can skip the Technical Stuff and sidebar text. (They'll still be there when you sneak up on them later.)

How This Book Is Organized

Building a dynamic, data-driven Web site is a process; certain things must be done in a certain order. (A simple example: If you want people to be able to set up accounts on your site and log in, first you have to create some means of storing data about users.) For that reason, this book is divided into parts and chapters that present information in the exactly the same order you need to follow when creating your own Web site. The following subsections tell you what to expect from those parts and chapters.

Part I: Planning a Web Site

Right off the bat, you need to decide whether to have your Web site support capabilities such as site membership, and whether to use the Master Pages feature of Visual Web Developer to give your site a consistent look and feel. This first part covers all the stuff you need to know if you want to build those features into your site.

Part II: Building Your Web Site

After you have the basic components for site membership and Master Pages in place, you can start focusing on specific content. In this part you'll discover the Visual Web Developer ways of defining content.

Part III: Personalization and Databases

Chances are, if you use Visual Web Developer to create your Web site, you'll want to offer more than just basic logins and simple static content. Part III covers topics you need to beef up your site with personalization, themes, and your own custom database tables.

Part IV: The Part of Tens

What *For Dummies* book would be complete without a Part of Tens? In this part you'll find a quick reference to the top ten buzzwords every VWD geek needs to know to get into the VWD Geek clubhouse, and resources you can access for information that goes beyond the scope of this book.

Icons Used in This Book

As you flip through this book, you'll notice little icons like these sprinkled about its pages. They point out little chunks of text that either deserve a little extra attention or (if they're obscure) deserve very little attention. For example, a Warning icon points out places where being careless could cause real problems, whereas a Technical Stuff icon points out facts nice to know but not super-important. The icons are



Tips point out handy tricks or techniques that can make things easier for you.



These icons point out techniques where you really need to watch what you're doing. The world won't end if you mess up, but fixing the problem won't be easy.



These icons point out tools and techniques that you'll use often in VWD, and hence should be high on your priority list of Things to Keep in Mind.



These icons point out text that isn't desperately relevant to all readers, though useful in an arcane way. You can skip these for now, if you like. They'll wait.

Where to Go from Here

There's a definite start-to-finish process to go through if you want to build a data-driven Web site. So if you're new to Visual Web Developer and are just starting your first site, starting at Chapter 1 is your best bet. Those of you who already have some experience with VWD and have already laid out some components of your sites can jump in anywhere.

Part I

Planning a Web Site

The 5th Wave By Rich Tennant



"Awww, cool - a Web Cam! You should point it at something interesting to watch. The fish bowl! The fish bowl!"

In this part . . .

Every project has to start somewhere. In Visual Web Developer, that usually means creating a new, empty Web site and configuring it to support membership. While you're at it, you'll need to get the hang of using the program's major features, and techniques for getting text and pictures into the pages you create. If you want to provide a consistent look and feel for all the pages in your site, you might want to consider creating a Master Page as well. All of those early steps are covered here in Part I.

Chapter 1

Getting Started

In This Chapter

- ▶ Getting your Web feet wet
 - ▶ Getting around in Visual Web Developer
 - ▶ Getting the help you need, when you need it
 - ▶ Being compatible
 - ▶ Finding someone to host your site
-

Visual Web Developer (VWD) is a tool for building dynamic, data-driven Web sites. The Express edition, which is the subject of this book, is specifically designed for non-professionals who want to learn to use VWD and related technologies without having to spring for the tools used by large corporations and professional programmers

That's not to say you can't create a "real" Web site with the Visual Web Developer Express. On the contrary, you can build a Web site of any complexity, and copy it to any Web server that supports ASP.NET 2.0 and other related technologies. It's just that the Express edition doesn't have some advanced features needed for very large commercial Web sites.

But, as a beginner, you're probably a long way from developing a large, busy commercial Web site. There's no need to spring for a more expensive version of the product until after you've mastered the Express edition.

Who VWD Is For

Visual Web Developer (VWD) is not a tool for computer beginners. Not by a long shot. VWD allows you to develop fancy Web sites by using existing Web technologies such as XHTML, XML, CSS, ASP.NET, as well as the .NET Framework 2.0, SQL Server, C#, and Visual Basic. In fact, most of the help and documentation for Visual Web Developer presumes that you're already familiar with those tools and technologies.

Of course, each said tool or technology is a book-length topic in itself. Total coverage of all that is beyond the scope of a single book written about Visual Web Developer. By way of a quick look, however, I describe what they are, how you use them, and where you find online resources with more information.

Installing Visual Web Developer

One of the biggest challenges beginners face when trying to use a large, complex program like Visual Web Developer is knowing *what* to do, *how* to do it, and *when* to do it. Technical documentation and “theory” don’t help with that. To really get your feet wet and understand the big picture, you need to spend some time using the program in a hands-on way.

Getting that hands-on experience is what this book is about. And to make sure you can get that experience, we’ve included a free copy of Visual Web Developer Express on the CD that comes with this book.

Like any program, you have to install Visual Web Developer Express before you can use it. So before you go any further here, take a moment to complete the installation instructions presented in Appendix B.

Getting Around in VWD

Once installed, starting Visual Web Developer is no different from starting any other program. Assuming VWD is properly installed on your computer, just click the Start button and choose All Programs⇨Visual Web Developer 2005 Express Edition. Figure 1-1 shows (roughly) how the program looks when it first opens. Don’t worry if yours looks different — it’s easy to move and size things to your liking.

The list given here briefly describes the purposes of the main panes pointed out in Figure 1-1. (If some of the terms are new to you, don’t worry about it; save the definitions for later when you start creating your site.)

- ✓ **Toolbox:** When you open a page or other item to edit, the Toolbox offers tools that allow you to add controls to the page.
- ✓ **Design Surface:** Also called the *design grid*, this is where you’ll create and edit your Web pages. Initially, you’ll see a Start Page here, which I’ll discuss that in a moment.

- ✔ **Solution Explorer/Database Explorer:** Each Web site you create is organized as a group of folders that shows up in the Solution Explorer. Any database you create for the site appears in the Database Explorer. Use the tabs at the bottom of the pane to switch between the two Explorer programs.
- ✔ **Properties:** Shows properties associated with the page or object with which you're currently working.

Using panes

You can move, size, show, and hide panes as needed to take advantage of your available screen space. To widen or narrow a pane, drag its innermost border left or right. If you have two or more panes stacked up along the edge of a screen, you can make the lower pane taller or shorter by dragging its top border up or down. The two-headed mouse pointers in Figure 1-2 show where you'd drag a couple of sample borders. (Ever see a two-headed mouse?)

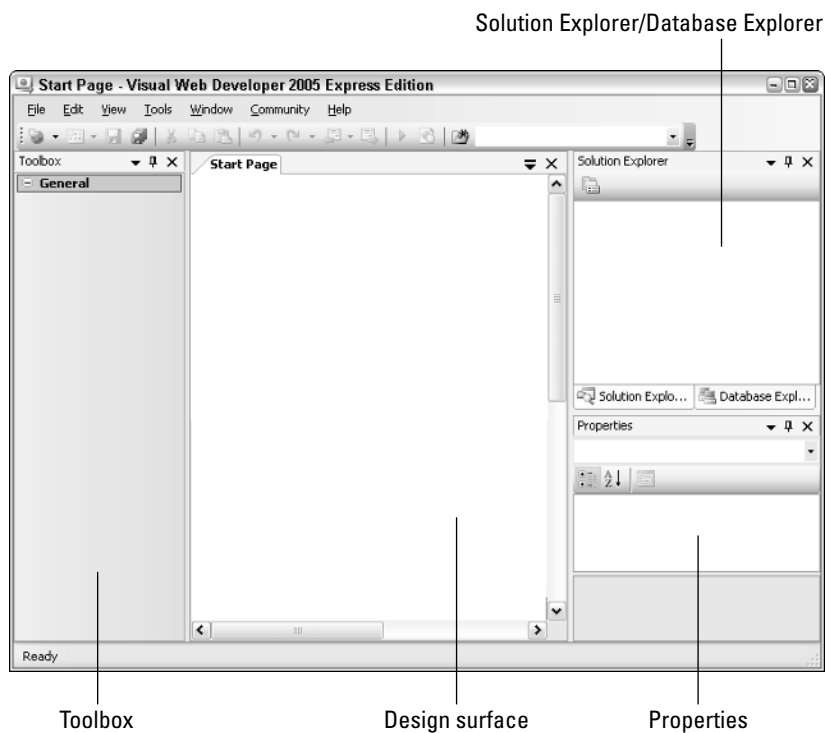


Figure 1-1:
VWD main
program
window.

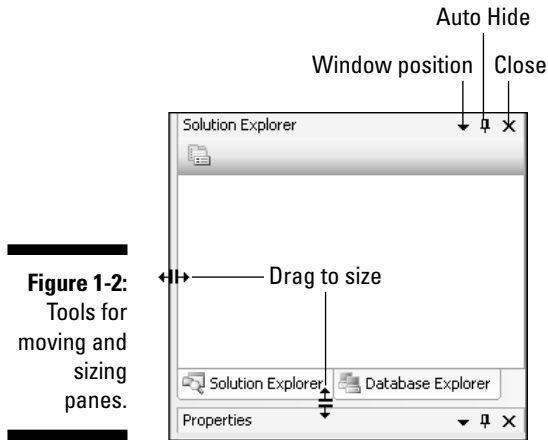


Figure 1-2:
Tools for
moving and
sizing
panes.

As pointed out in Figure 1-2, the title bar of a pane contains three buttons titled Window Position, Auto Hide, and Close. Clicking the Window Position option gives you the following choices on a drop-down menu:

- ✓ **Floating:** Converts the pane to a free-floating window that you can move and size independently of the program window.
- ✓ **Dockable:** Docks a pane that is currently showing as a tabbed document.
- ✓ **Tabbed Document:** Moves the pane into the Editing area, identified by a tab at the top of the area. Click the tab to make the pane visible. Right-click the tab and choose Dockable to re-dock the pane to the program window.
- ✓ **Auto Hide:** Converts open panes to hidden panes along the border of the program window, as in the example shown in Figure 1-3. To bring a pane out of hiding, point to (or click) its name.
- ✓ **Hide:** Hides a pane immediately so only its name appears along the border. To bring the pane out of hiding, click (or just point to) its name.



To quickly put all of the visible panes into hiding, choose Window⇨Auto Hide All.

When you bring a hidden pane out of hiding, you'll notice that the Auto Hide "pushpin" is horizontal. Clicking that pushpin keeps the pane open.

Getting panes back to normal

With so many optional panes, and so many ways to move and size things, it's easy to make a real mess of your program window. But don't worry; to whip everything back into shape, all you have to do is choose Window⇨Reset Window Layout from the menu bar.

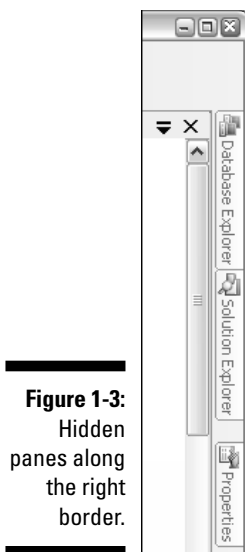


Figure 1-3:
Hidden
panes along
the right
border.

Don't forget the View menu

The View option in the menu bar, shown in Figure 1-4, provides access to all optional panes (also called *windows* because they can be free-floating). If you close a pane by clicking its Close (X) button, you can always bring the pane back into view by choosing its name from the View menu.

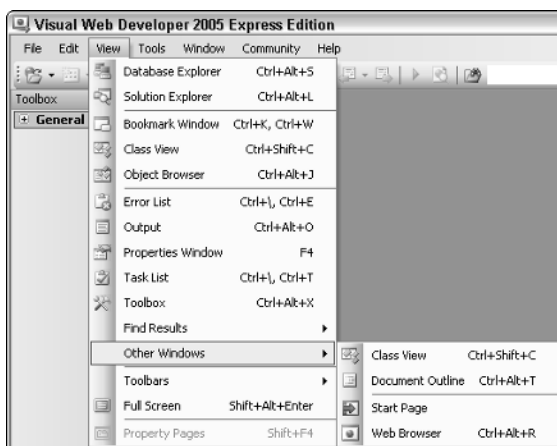


Figure 1-4:
The View
menu.

Some options on the View menu, like Object Browser and Error List, won't play any significant role until you start building your Web site. In most cases, these panes appear automatically when needed. I'll discuss those other panes as the need arises in this, and later, chapters.

The View menu also offers a Toolbars option you can use to show and hide various toolbars. As with many of the optional panes, toolbars appear — and disappear — as appropriate to whatever task you're performing at the moment. So don't fret about which toolbars are (or aren't) visible right now.

About the Start Page

The Start Page, shown in Figure 1-5, appears each time you open VWD. Under Recent Projects you'll see a list of Web sites you've worked on recently (if any). To open one of those Web sites, just click its name.

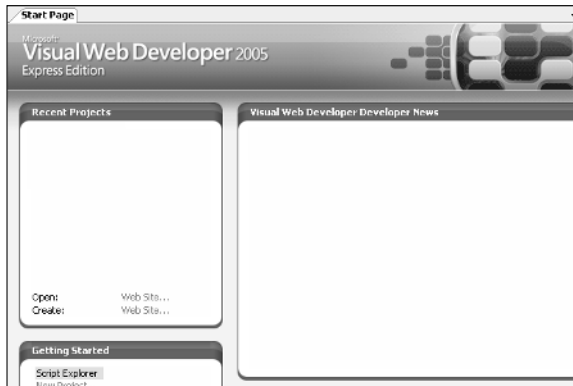


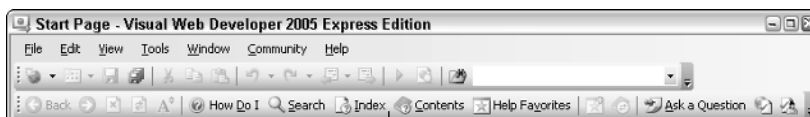
Figure 1-5:
The VWD
Start Page.

The Start page doesn't contain anything that's required to build a Web site. In fact, after you've created or opened a Web site, you can close the Start page by clicking the Close (X) button in its upper-right corner. If you change your mind and want to bring the Start page back to the screen, just choose View⇨ Other Windows⇨ Start Page from the menu.

Using VWD Help

Quick, easy access to information is key to using VWD well. There's too much information for anyone to memorize. Visual Web Developer offers many ways of getting the information you need when you need it. The Help command on the menu bar, and the Help toolbar shown at the bottom of Figure 1-6, provide many ways to look up information, as summarized below.

Figure 1-6:
The Help
toolbar
(bottom).



Help toolbar



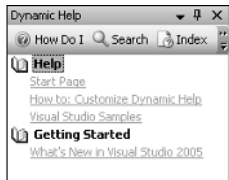
If the Help toolbar isn't visible, choose View⇨Toolbars⇨Help from the menu bar. The navigation buttons at the left side of the Help toolbar will be enabled when you have some Help content visible on your screen.

It's probably no stretch to assume you can find your way around the Help system and figure things out from the options available to you. But just so you know what's available, I'll briefly summarize the main Help options:

- ✓ **How Do I:** Opens the "How Do I?" page in the Design Surface. The page contains links to topics that describe how to perform various common tasks in VWD.
- ✓ **Search:** Provides many options for searching both local and online help for a specific word or phrase.
- ✓ **Index:** Provides an index, like the index at the back of a book, where you can look up a term alphabetically.
- ✓ **Contents:** Opens the Help Table of Contents in the right pane. Use it as you would the Table of Contents at the start of a book.
- ✓ **Help Favorites:** Opens the Help Favorites pane at the right side of the program window. When you're viewing a Help page, you click the "Add to Help Favorites Button" in the Help toolbar (just to the right of the Help Favorites button) to add the current page to your Help Favorites.
- ✓ **Dynamic Help:** (Help menu only) Opens the Dynamic Help pane in the lower-right corner of the screen (Figure 1-7). As you create a page and click different types of controls, links to information about the context in which you're working automatically appear in this pane.
- ✓ **Help on Help:** (Help menu only) Offers detailed information on all the built-in help available to you.

If you're new to Web development, much of the help may be over your head and not very helpful at all. Try not to let that intimidate you. Everyone has to be a beginner at some point. A major goal of this book is to get you from that absolute-beginner point to a more experienced level where the technical documentation can actually be helpful.

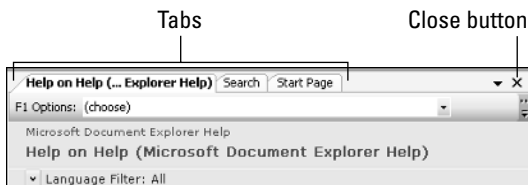
Figure 1-7:
The
Dynamic
Help pane.



Closing Help pages and panes

Most Help pages open up in the Design surface. You can switch among open pages using the tabs at the top of the surface. To close a pane, click its tab and then click the Close (X) button at the top of the Design surface (Figure 1-8). Or right-click the tab and then choose Hide.

Figure 1-8:
Tabs and
Close
button.



Panes, like the Dynamic Help pane shown back in Figure 1-7, can be handled like any other pane, using tools and techniques described near Figure 1-2.

Online resources

No matter what your level of expertise coming into this book, sometimes you need specific information about the technologies that VWD supports. That includes the .NET Framework 2.0, ASP.NET, CSS, HTML, XML, SQL Server 2005, and the C# programming language. You don't need to master all these topics right off the bat. Heck, the printed documentation for the .NET Framework alone is over 8,000 pages. Not many people will be interested in learning or using everything it has to offer. It's just too darn much information, most of which has nothing to do with building Web sites.

The trick is being able to find the information you need when you need it. Certainly the Help resources described in the previous sections have much to offer. But it never hurts to have a few extra resources at your fingertips.

A good first resource is the Visual Web Developer section of my own personal Web site at www.coolnerds.com. (You can browse straight to that section using www.coolnerds.com/vwd). For more technical information, consider the following Web sites:

- ✓ **.NET Framework Developer Center:** <http://msdn.microsoft.com/netframework/>
- ✓ **ASP.NET QuickStart Tutorials:** www.asp.net/tutorials/quickstart.aspx
- ✓ **Cascading Style Sheets (CSS) — W3C:** www.w3.org/Style/CSS/
- ✓ **SQL Server Developer Center:** <http://msdn.microsoft.com/SQL/>
- ✓ **Visual C# Developer Center:** <http://msdn.microsoft.com/vcsharp/>
- ✓ **XHTML Home Page:** www.w3.org/TR/xhtml1/
- ✓ **XML (Extensible Markup Language):** www.w3.org/XML/

Being Compatible with Web Browsers

Every Web author has to make a trade-off decision between Web browser compatibility and fancy features. If you want to ensure that virtually everyone can visit your site, then you want to be compatible with very early versions of Web browsers — say, Internet Explorer 3 and Netscape Navigator 3. However, those older browsers don't support the better, fancier stuff you can use with modern Web browsers.

If you want to use the capabilities of modern browsers, you have to limit your Web site to using only those. This is not as big a sacrifice as it might seem; almost everyone has more recent browsers. Few sites gear their new content to version 4 and earlier browsers anymore, and most browser manufacturers are keeping up with current XHTML specifications. And since XHTML is the future for browsers anyway, most Web authors lean toward those specifications.

Anyway, I'm sure one could debate browser compatibility ad infinitum — but here's the bottom line: Make that decision early on so you're better prepared to create consistent pages for the site and match what your visitors are most likely using. You use the Options dialog box in VWD to set browser compatibility; here are the steps:

- 1. Choose Tools⇨Options from VWD's menu bar.**

The Options dialog box opens.

- 2. Click the + sign (if any) next to Text Editor HTML.**

- 3. Click Validation.**

- 4. Choose your preferred browser compatibility from the Target drop-down list.**

In Figure 1-9, I chose "XHTML 1.0 Transitional (Netscape 7, Opera 7, Internet Explorer 6)."

- 5. Click OK.**

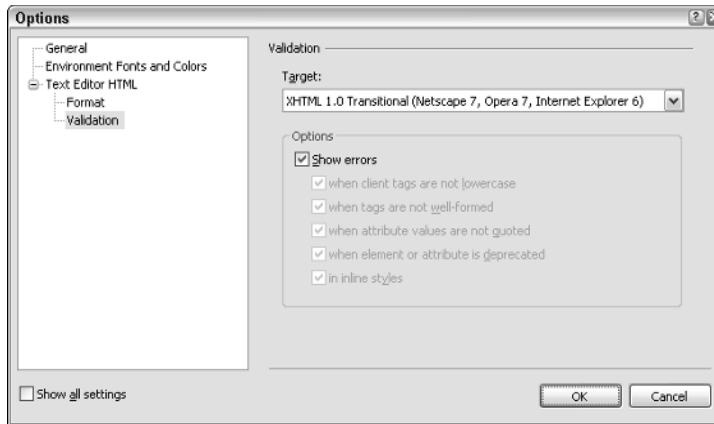


Figure 1-9:
Choosing
target
browsers.



In case you're wondering why the option in the Options dialog box is named Validation, it's because VWD automatically *validates* your page each time you open it — that is, it makes sure everything in the page works properly when a visitor opens the page from a Web browser. If VWD finds a problem, it alerts you via an error message.

The Options dialog box has a whole slew of other options. You can see 'em if you choose the Show All Settings check box at the bottom of the dialog box. There are a ton of other options to choose from, but the only one worth bothering with at the moment would likely be the General tab under HTML Designer.

As you'll discover in Chapter 2, you can edit pages either in a WYSIWYG graphical view, or a more textual Source view. You can switch views at any time with a single mouse click, which is no big deal. But if you choose Design View rather than Source View from the HTML Designer General options, as in Figure 1-10, your pages will initially open in Design view.

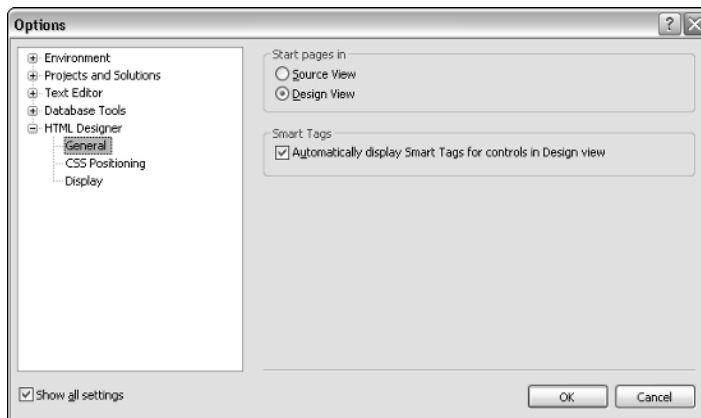


Figure 1-10:
Choosing to
open pages
in Design
view.

When you've finished making your choices in the Options dialog box, just click OK to save your choices and return to VWD.

Publishing Your Web Site

As you may already know, simply creating a Web site on your own PC is only a first step; you can admire it while it sits there, but that doesn't make your site available to the public at large. That can happen only after you've obtained a domain name and published your site to a Web server located at that domain name.

The company that provides the space on which you publish your site is often referred to as a *hosting service*, a *hosting provider*, a *Web presence provider*, or even a *WPP* for short. The hosting services that specifically support the technologies you use in VWD to develop your Web site are *ASP.NET 2.0 Hosters*.

Eventually you'll need a hosting service that supports ASP.NET 2.0 and SQL Server 2005. You can find a list of such hosting services at www.asp.net/hosters/. There's no reason to sign up right this minute, especially if your site isn't built yet. But you can certainly shop around as time permits.

Chapter 2

Creating a Web Site

In This Chapter

- ▶ Creating a new Web site
 - ▶ Defining and using folders
 - ▶ Creating and editing Web pages
 - ▶ Viewing pages in a Web browser
-

The first step to using Visual Web Developer is to create a new, empty Web site. The phrase *Web site* in this context does not mean a Web site that people can browse to on the Web — that would be a *live Web site* or a *production Web site*. The Web site you create is initially just a bunch of files and folders on your computer that nobody except yourself can get to.

After you've created your new Web site, you can then start designing and creating the site. Doing so involves creating folders and pages, putting things on pages, seeing how pages look when viewed in a Web browser, and so forth. Such things fall under the heading of “everyday basic skills” because you'll do them every time you use Visual Web Developer. They are also the topic of this chapter.

Creating a Web Site

From the standpoint of Visual Web Developer, a *Web site* is basically a folder that contains still more folders and all the files that make up a single, complete Web site. Unlike a live Web site, the empty one you create will be in your *file system*, or, in less technical terms, your own computer's hard disk, where nobody except you can get to it.

Even in Visual Web Developer, there will be times when you may have to write some code, in a programming language, to get a job done. Visual Web Developer supports several programming languages — including Visual Basic, C#, and J#. In this book, we use C# (pronounced *see sharp*). If you're already familiar with one of the other languages, you're welcome to use it

instead. But given that you probably won't be writing much code, and given that these languages are so similar, you shouldn't have any problems using C# even if you're not familiar with the language.

To create a new Web site that uses C# as its default programming language, follow these steps:

1. From VWD's menu bar, choose **File** → **New Web Site**.

The New Web Site dialog box (shown in Figure 2-1) appears.

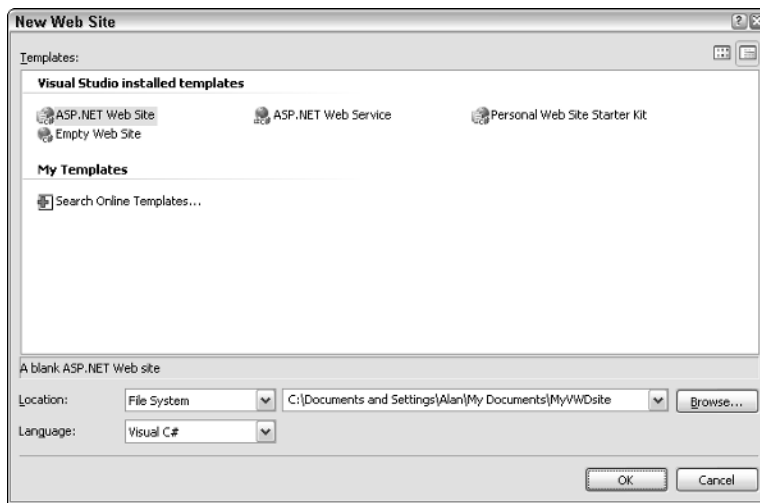


Figure 2-1:
The New
Web Site
dialog box.

2. Click **ASP.NET Web Site**.

3. From the **Location** drop-down list, choose **File System**.

4. From the **Language** drop-down list, choose a preferred programming language.

As mentioned, I use Visual C# throughout this book.

5. Optionally, change the location and name of the Web site.

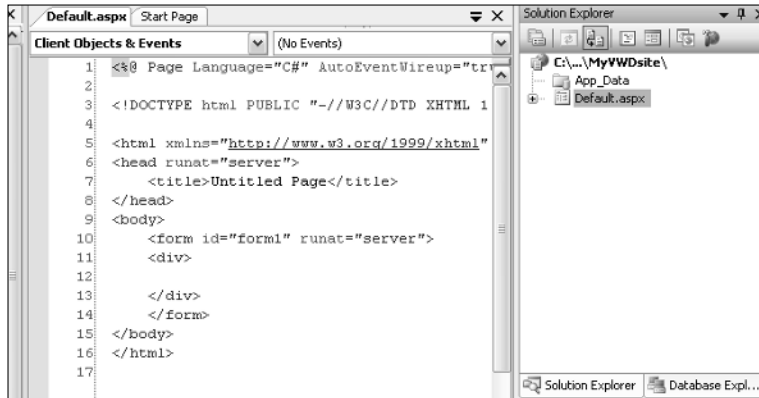
The site name is at the end of the lengthy path to the right of the Location drop-down list. In the example shown in Figure 2-1, I've opted to create a site named `MyVWDsite` in the My Documents folder.

6. Click **OK**.

It takes a few seconds for VWD to create the site. When the site is ready, you end up with an empty page named `Default.aspx`, which is the Web site's home page. You also get a folder named `App_Data`, which is where your site's database will be stored (eventually). These are both listed under the site's root folder in Solution Explorer at the right side of the program window.

The `Default.aspx` page also opens automatically in the Design surface. You'll see its name in a tab at the top of the Design surface. In Design view, the page just looks like a big white sheet of paper. In Source view, you'll see some HTML and other stuff that isn't visible to people who visit the site, as in Figure 2-2.

Figure 2-2:
A new Web site as shown in Solution Explorer, and an open page in Source view.



Speaking of Design view and Source view, notice the two little buttons titled Source and Design near the bottom of the window (Figure 2-3). Use those buttons to switch from one view to the other. Note that these are just two different ways of looking at the page, as follows:

- ✓ **Design:** In this view, the page looks much as it will in a Web browser. Use this view for normal WYSIWYG (what you see is what you get) editing.
- ✓ **Source:** This view shows the HTML and other tags for the page — “instructions” that tell the page how to behave and how to look in a Web browser.

Figure 2-3:
The Design and Source buttons.



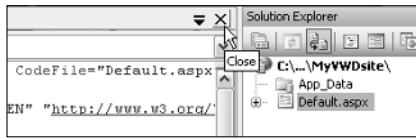
Closing and Opening Pages

As mentioned, when you create a new Web site, VWD automatically creates one blank page, named `Default.aspx`, for the site. Your site will likely contain many pages. Typically you only want to work on one page at a time

(or perhaps a few pages) — and that means opening and closing pages for editing. Right now there's only one page in the site, and it's open. To close a page:

- ✓ Click the Close (X) button in the upper-right corner of the Design surface (near the mouse pointer in Figure 2-4).
- ✓ Or choose File ⇨ Close from the menu bar.

Figure 2-4:
Close
button for
Default.
aspx.



If you made any changes to the page since you last opened it, a dialog box asks if you want to save those changes. Choose Yes (unless you made a mess of things and don't want to save your changes). The page disappears from the Design surface, but its name remains visible in Solution Explorer.

To open a page, just double-click its name in Solution Explorer. When the page is open, use the Design and Source buttons to choose how you want to view the page.

Creating and Using Folders

You can use folders to organize pages and other components in your Web site in much the same way you use folders in Windows to organize files. For example, you might want to create a folder for storing all the site's pictures. To create a folder, follow these steps:

- 1. Make sure your site is open and go to the Solution Explorer pane.**
- 2. Right-click the site name at the top of the Solution Explorer tree and choose New Folder, as shown in Figure 2-5.**
- 3. Type in a new name for the folder, and then press Enter.**

Figure 2-6 shows an example I created — a new folder named `Images`. I'll use that folder to store some of the pictures for my Web site.



To rename a folder, right-click its icon in Solution Explorer, choose Rename, type in the new name, and press Enter.

Figure 2-5:
Creating a
folder.

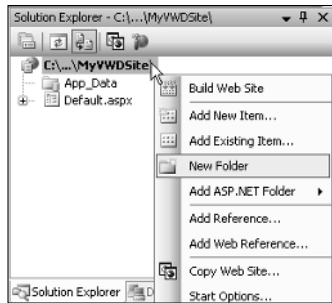
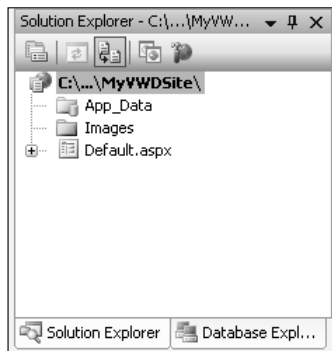


Figure 2-6:
A new
regular
folder
named
Images.



Copying files to folders

You can create pages in any folder you wish within your site. But what about files you may have already created, such as pictures you intend to use on your site? Well, you can move or copy those from their current location into folders in Solution Explorer, using the drag-and-drop method.

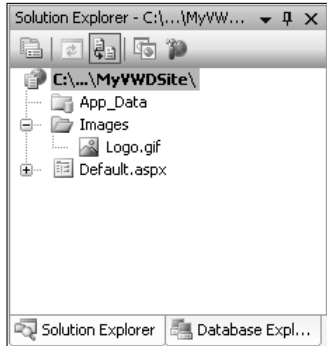
For example, to copy a picture from your My Pictures folder into a folder, leave VWD open with your site folder open in Solution Explorer. Then just drag-and-drop any icon, or selected group of icons, from your My Pictures folder to the site folder in Solution Explorer.

Figure 2-7 shows an example where I dragged a file named `logo.gif` from the My Pictures folder to the Images folder in Solution Explorer. When the Images folder is expanded (showing a - mark instead of a + mark), the `logo.gif` file within that folder is visible.



Use the + and – keys next to the folder name to show or hide the contents of the folder.

Figure 2-7:
Here's the
Logo.gif file
in the
Images
folder.



If drag-and-drop isn't your thing, you can copy and paste instead. For example, if you have some pictures in a folder that you need to use in a Web site, open that folder in Windows. To copy multiple pictures, select their icons using standard Windows techniques. Then right-click the picture's icon (or any selected icon) and choose Copy. Then, in Solution Explorer, right-click the name of the folder into which you want to place the picture(s) and choose Paste.

Renaming and deleting folders

You can rename or delete any regular folder by using techniques similar to those in Windows:

- ✓ To rename a folder, right-click the folder, choose Rename, enter the new name, and press Enter.
- ✓ To delete a folder, right-click the folder name and choose Delete.

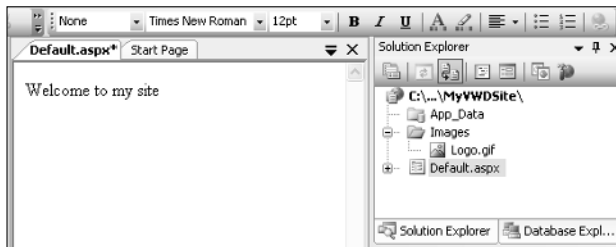
Editing Pages

To edit an existing page in VWD, you first need to open the page so it's visible on the Design surface. To open a page, double-click its icon in Solution Explorer. When you open an .aspx page (such as `Default.aspx`), you see the Design and Source buttons at the bottom of the Design surface, so you can switch between the two views.

Adding text to a page

As mentioned briefly in the preceding section, VWD allows you to edit an .aspx page in either Design view (which shows what the page looks like in a browser) or Source view (raw HTML and ASP.NET). Typing and editing text in Design view is like typing and editing in Microsoft Word, FrontPage, or just about any other text editor or word processor: You click the page to get the cursor into position, and then type your text. Figure 2-8 shows an example where I typed the text `Welcome to my site` on the (otherwise empty) `Default.aspx` page.

Figure 2-8:
Here's
some text
added to
`Default.aspx`.



All the standard text-editing tools and techniques work on the Design surface. For example, you can delete with the Backspace and Delete keys. You can select text by dragging the mouse pointer through the text, or by holding down the Shift key while positioning the cursor with the navigation keys. You can copy and paste text to or from the Design surface.

Selecting and formatting text

Selecting and formatting text works the same in VWD as it does in word processing programs. To format a chunk of text, first *select* (highlight) the text you want to format by dragging the mouse pointer through that text. Then choose a format option from the Formatting toolbar.

In Figure 2-9, for example, I selected the text `Welcome to my site` so it's highlighted. Then I clicked the Block Format button at the left side of the Formatting toolbar; the figure shows how the screen looks just before I click the `Heading1 <H1>` option.

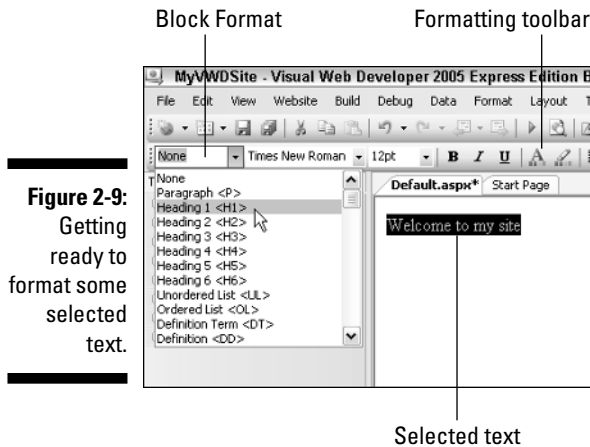


Figure 2-9:
Getting
ready to
format some
selected
text.

Undoing changes

As in most modern applications, you can undo the most recent changes you made to a page by using either of these methods:

- ✓ Choose Edit ⇨ Undo from the menu bar.
- ✓ Press Ctrl+Z.
- ✓ Click the Undo button in the Standard toolbar.

VWD has multiple levels of undo and redo, so you're not limited to undoing only your most recent change. However, when you save a page, you "commit" all changes up to that point. Undo actually only reaches back as far as your last save.

Adding pictures

To add a picture to a page, first make sure you move or copy the original picture into a folder in Solution Explorer. Make sure all the files that make up your Web site — including pictures — are in folders within Solution Explorer. Otherwise, when you upload your finished site to a Web server, the pictures won't be included in the upload, which means that anyone trying to view a page that contains a picture will just see a red X where the picture should be.

To add a picture to a page, just drag its icon from its folder in Solution Explorer onto the page. Figure 2-10 shows an example where I dragged a picture named `Logo.gif` onto the page, from a folder named `Images` in Solution Explorer.

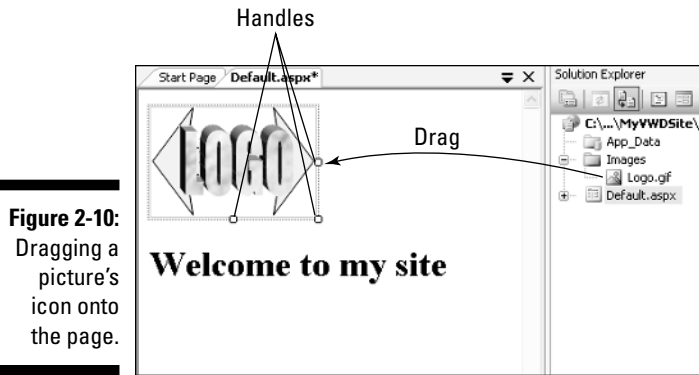


Figure 2-10:
Dragging a
picture's
icon onto
the page.

On the Design surface, the picture shows dragging *handles* (small squares along the border). If you don't see the handles, click the picture to select it; they'll show up. Then you can size the image by dragging any handle.

Changing properties

If you look around the room you're in right now, you'll probably see many physical objects — PC, keyboard, mouse, desk, and whatever else is in the room. No two objects are exactly alike. Instead each object has certain properties such as size, shape, weight, purpose, and so forth that makes it unique.

Just about anything you add to a Web page is also an *object*. And (like objects in the real world) objects on Web pages have properties. An object's *properties* are settings that define its characteristics — such as size, shape, location on the page, and so forth.

To see, and perhaps change, an object's properties, just select (click) the object of interest and look at the Properties pane. Or, right-click the object and choose Properties. The item's properties appear in its *properties sheet*, which always shows in Visual Web Developer's Properties pane.

Figure 2-11 shows an example where I'm viewing the properties for a picture. The `` tag near the top of the Properties sheet is a reflection of the fact that, like all pictures in all Web sites, this particular picture is displayed by an HTML `` tag.



To make the Properties sheet free-floating (as in Figure 2-11), choose Floating from its Window Position button. (See Chapter 1 for more details.)

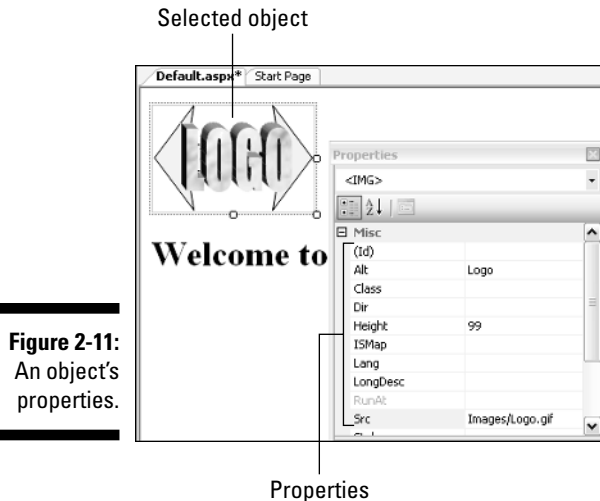


Figure 2-11:
An object's
properties.



HTML (Hypertext Markup Language) is a set of tags used to define the general format of items on a Web page. Visual Web Developer is a tool for people who already know that, and know what the HTML tags are. That's why when you click on a picture and look at its Properties sheet, you see the term `` in the properties sheet rather than the word *picture*. The assumption is you already know that `` tags show pictures.

Different types of objects have different properties. The list of properties for an object may be long, so you may need to use the scrollbar at the right side of the list to see them all. Most properties can be changed by clicking the column to the right of the property name.

An object's properties sheet provides *a* means of tweaking optional settings for the object, but not the *only* means. In fact, any settings that relate to the look and feel of the object on the page are best dealt with outside the properties sheet through the Style Builder or CSS (Cascading Style Sheets), which are covered in Chapters 5 and 6.



To get to the Style Builder, right-click the item you want to style and choose Style.

Switching views

The Design view shown in the previous figures allows you to edit a Web page in a WYSIWYG (pronounced *wizzy-wig* — “what you see is what you get”) mode. In other words, what you see in the Design view is very similar to what a person visiting the page with a Web browser would see. That's the way most people like to work.

As mentioned earlier, there's also a Source view for editing Web pages. To switch to Source view, click the Source button at the bottom of the Design surface. (To switch back to Design view, click the Design button at the bottom of the Design surface.)

The Source view shows the HTML tags (and other stuff) that VWD is creating behind the scenes as you create the page in the Design view. Whether or not any of that looks familiar depends on your familiarity with HTML.

Those of you who are familiar with HTML will recognize similarities between the tags and the content of the page. For example, if you switch to Source view while viewing the page in Figure 2-11, you might recognize the tags shown here:

```

<h1>Welcome to my site</h1>
```

Dragging the `logo.gif` file onto the page created the `` tag. `Welcome to my site` is typed text. The `<H1>` and `</H1>` tags were added by selecting the typed text and choosing *Heading 1 <H1>* from the Block Format menu on the toolbar.

There's rarely any need to work directly with HTML tags. So don't get too uncomfortable looking at all the gibberish in the Source view. But for those of you who are familiar with HTML, I offer the following section.

Editing in Source view

If you're familiar enough with HTML to work directly in the Source view, you'll be able to take advantage of VWD's *IntelliSense* technology. IntelliSense "looks at" what you're typing, or have already typed, and provides menus of options representing valid HTML keywords relevant to the context in which you're typing. I think an example will best illustrate.

Suppose you're working in the Source view and want to insert an HTML tag. As soon as you type the opening angle bracket, `<`, a menu appears listing valid words you can type after the opening bracket, as in Figure 2-12.

When the IntelliSense menu appears, you have two choices. You can ignore it and just keep typing. Or you can scroll through the menu and double-click the word you want to insert next into the brackets.

If you enter the first tag of a pair, IntelliSense will usually add the closing tag for you automatically. For example, if you type `<p>` into the page in Source view, you'll get `<p></p>`. The cursor lands between the two tags so you can type within the tags immediately.

Figure 2-12:
IntelliSense
menu in
Source
view.



When you use XHTML for validation, unpaired tags such as `` and `
` must end with a slash and a closing angle bracket (`/>`). For example, `` and `
`. When you edit in the Design surface, the correct tags are created automatically. If you plan on writing HTML yourself, make sure you're up on XHTML rules, which you can find online at www.w3.org/Markup.

If you're not familiar with HTML and don't quite get what value IntelliSense offers, don't worry about it. For the most part, you can create Web sites in VWD by using simple drag-and-drop techniques and properties without typing any HTML at all.



All Web pages contain some HTML, even though you never see HTML tags in pages. That's because your browser *renders* the HTML into what you see in your browser. For example, the HTML `Hello` renders as the word **Hello** in boldface, without the tags. If you use Internet Explorer as your Web browser, you can choose View \rightarrow Source from its menu bar to see the unrendered HTML source page.

Saving your work

As soon as you start editing a page, you'll notice that the tab that shows the page name at the top of the Design surface is boldfaced and shows an asterisk, as in the example shown in Figure 2-13. The asterisk means "you've changed this page since you last saved it, and those changes have not yet been saved." To save the page with your most recent edits, use whichever technique below is most convenient:

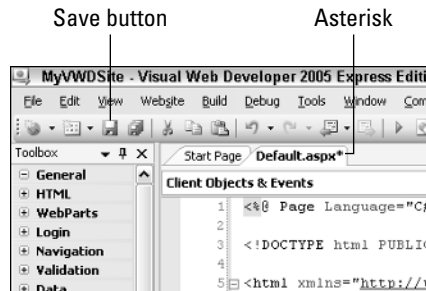
- ✓ Click the Save button in the toolbar (shown near the mouse pointer in Figure 2-13).
- ✓ Press Ctrl+S.
- ✓ Choose File \rightarrow Save *Pagename* from the menu bar (where *Pagename* is the name of the page you're editing).



Clicking the Save All button next to the Save button will do the trick too. As its name indicates, the Save All button saves all open pages.

If you attempt to close an edited page without saving it first, you'll see a prompt asking whether you want to save your changes. You should choose Yes unless you're sure you want to abandon all changes you've made since you last saved the page.

Figure 2-13:
Changed
page and
Save button.



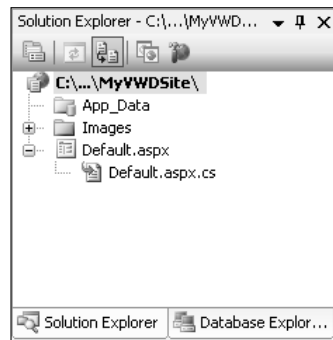
Dealing with code-behind files



Many .aspx pages have a corresponding *code-behind* file. These pages contain programming code that defines the behavior of the page, as opposed to any kind of visible content. The code in a code-behind page is written in whatever programming language you choose when first creating the Web site. In this book's examples, that will always be the C# programming language.

In Solution Explorer, any page that has a code-behind file shows a + sign next to its icon, or a minus sign with an icon for the code-behind file. The name of the code-behind file is the same as the name of the .aspx page with a .cs extension added on, as in Figure 2-14. (The .cs is for C#. If you use a different programming language, the extension will adjust accordingly, for example, .vb for Visual Basic.)

Figure 2-14:
Icon for
code-
behind file
under
Default.
aspx.



When you open a code-behind file you see *code*. There is no Design view for code, because a code-behind file can only contain code — computer instructions written in a programming language like C# or Visual Basic. The code may look something like this:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

The meaning of the code isn't important at the moment. Suffice it to say that the code file is where the *logic* of a Web page resides, whereas HTML controls what *appears* on-screen when someone views the page through a Web browser. You'll see examples of how that works in Chapter 9. Whether or not you'll ever have to deal with code-behind pages depends on what you want your Web site to do — so don't let all that gibberish in the code-behind page worry you.

To close a code-behind page, just click its Close (X) button near the upper-right corner of the Design surface.

Titling Pages

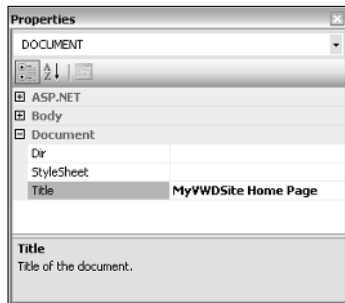
Every Web page has a page title that appears in the Web browser's title bar when someone is viewing the page. That same title also shows up in links to the page from most search engines, like Google. In HTML, the title must be placed between `<title>...</title>` tags, which in turn must be inside the page's `<head>...</head>` tags.

In Visual Web Developer, you can follow these steps to create a title for whatever page you're currently editing in the Design surface:

1. From the drop-down list at the top of the Properties sheet, choose <DOCUMENT>.
2. Scroll to the bottom of the Properties sheet and type your page title as the Title property.

For example, in Figure 2-15, I've given the page the general title *MyVWDSite Home Page*.

Figure 2-15:
Typing a
page title
for the
browser's
title bar.



You won't see anything in the Design view of the page, because this title doesn't show up on the Web page. In Source view, you'll see the HTML required to show the page title:

```
<title>MyVWDSite Home Page</title>
```

The only other time you'll see that title is when you view the page in a Web browser, where it appears in the title bar at the top of the Web browser's program window. Which brings us to . . .

Viewing Pages in a Web Browser

The Design view of a page gives you a good sense of how the page will look in a Web browser. But it doesn't always provide an exact view. To put your page to a real test, view the page in a Web browser. After all, when the site is up and running on a Web server, everyone who visits the site will be doing so through a Web browser.

To view, in a Web browser, the page you're currently working on in Design or Source view in VWD, use whichever method below is most convenient for you:

- ✓ Right-click some empty space on the page and choose View In Browser.
- ✓ Click the View in Browser button in the toolbar (left side of Figure 2-16).
- ✓ Press Ctrl+F5.

If you want to view in a Web browser a page that isn't currently open, right-click its name in Solution Explorer and choose View in Browser, as shown at the right side of Figure 2-16.

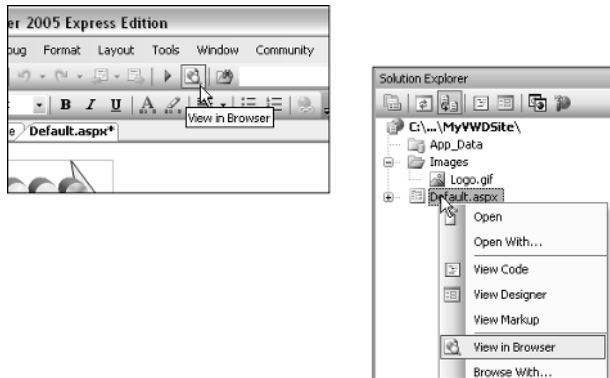


Figure 2-16:
Two ways to
view a page
in a Web
browser.

The page opens in a Web browser, mostly likely Microsoft Internet Explorer. Figure 2-17 shows the `Default.aspx` page open in Internet Explorer. Note the page title, `MyVWDSite Home Page`, in the upper-left corner of the figure. That's the only place that page title is actually visible on the screen.

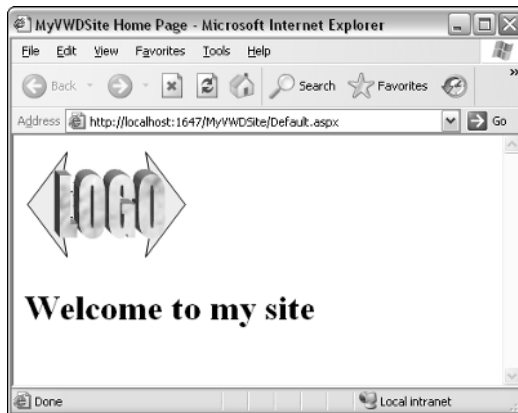


Figure 2-17:
The
Default.
aspx page
in Microsoft
Internet
Explorer.

The browser window will likely cover Visual Web Developer's program window. To go back to designing your page, just close the Web browser by clicking the Close (X) button in its upper-right corner.

Opening and Closing Web Sites

Creating a Web site is no small feat, and it's pretty unusual to accomplish much in a single sitting. So before I end this chapter, let's take a quick look at managing Web sites as a whole.

When you're in Visual Web Developer and working on a site, you can close the site without closing the program — handy if you're working on multiple sites. To close the current Web site, choose File⇨Close Project from VWD's menu bar. (VWD often uses *project* to mean *Web site*.)

When you first open VWD, you won't be taken to your Web site automatically. Even so, you can easily open your site by any of the following methods:

- ✓ Click your site's name under Recent Projects on the VWD Start Page.
- ✓ Choose File⇨Recent Projects from the menu bar. Then click the name of the Web site you want to open.
- ✓ Choose File⇨Open Web Site from the menu bar. When you choose this option, the Open Web Site dialog box opens. Click File System, then use the directory tree to navigate to the folder in which you placed your project (look for a regular folder icon with whatever name you entered when you created the site).

Whichever method you use to open your Web site, the folders and files appear in Solution Explorer exactly as you left them.

Chapter 3

Configuring a Membership Site

In This Chapter

- ▶ Configure your Web site for membership
 - ▶ Create folders for members-only content
 - ▶ Prevent anonymous users from accessing members-only content
 - ▶ Create and manage user accounts
-

One of Visual Web Developer's best features is its ability to create a membership Web site with minimal fuss and muss. As the owner of a membership Web site, you can control who has access to what content. For example, you can have general content for *anonymous users*, people who just happen to wander into the site, and then you can have premium content for members only, where *members* are people who have set up an account on your site.

The basic idea is pretty simple. You create a folder, perhaps named MemberPages, where you put all members-only content. Then you set up a *role*, perhaps named SiteMembers. Finally, you create a rule that says "anonymous users cannot access content in the MemberPages folder; only people in the SiteMember role can access pages in the MemberPages folder." In other words, if someone who just happens to visit your site wants to see your special content, that person must first join your site by setting up a user account.

Creating a Folder for Members-Only Content

The first step to setting up a membership site is to decide how you're going to organize your content. You'll likely want some of your site's content to be available to *anonymous users*. An anonymous user is anyone who visits the site without creating or logging into an account on your site.

In addition to the general content that's available to everyone, you may want some privileged members-only content that's available only to site members — people who have joined your site by setting up a user account.

To keep the privileged members-only content separate from the general content, you need a folder, perhaps named `MemberPages`. Keep all your privileged content in the `MemberPages` folder. Keep general content out of that folder. The `MemberPages` folder can be just a regular folder in which you store pages.

To create a folder in Visual Web Developer, right-click the site name at the top of Solution Explorer and choose `New Folder`. Then type the folder name and press `Enter`. Figure 3-1 shows an example where I've added a new, regular folder named `MemberPages`. As its name implies, the `MemberPages` folder will (eventually) contain pages that only site members can access.

Figure 3-1:
A regular folder named `MemberPages` has been added to the site.



There is nothing special about the `MemberPages` folder. It's just an empty folder in which you can store pages. If you want to use that `MemberPages` folder to store privileged members-only content, you need some means of distinguishing between site members and the general riff-raff (anonymous users). In other words, your site needs an infrastructure that can store information about site members, allow site members to log into their user accounts, and so forth. To create that infrastructure, you use the Web Site Administration Tool.

Using the Web Site Administration Tool

The Web Site Administration tool, often abbreviated WAT, is the tool you use to administer access to pages within your Web site. To start the tool, follow these steps:

- 1. If you haven't already done so, start Visual Web Developer and open your Web site.**
- 2. Choose `Website` → `ASP.NET Configuration` from the menu bar.**

The tool opens in a Web browser like the one in Figure 3-2.

The tabs near the top of the page (Home, Security, Application, and Provider), and links on the Home tab, provide tools both for creating a membership site and for managing a membership site after it's created. The first step to setting up a membership site is to make sure Visual Web Developer can connect to SQL Server, the database program used to store information about users. To test your connection, follow these steps:

1. Click the **Provider** tab or the **Provider Configuration** option on the **Home** tab.
2. Choose the **Select A Single Provider For All Site Management Data** option.

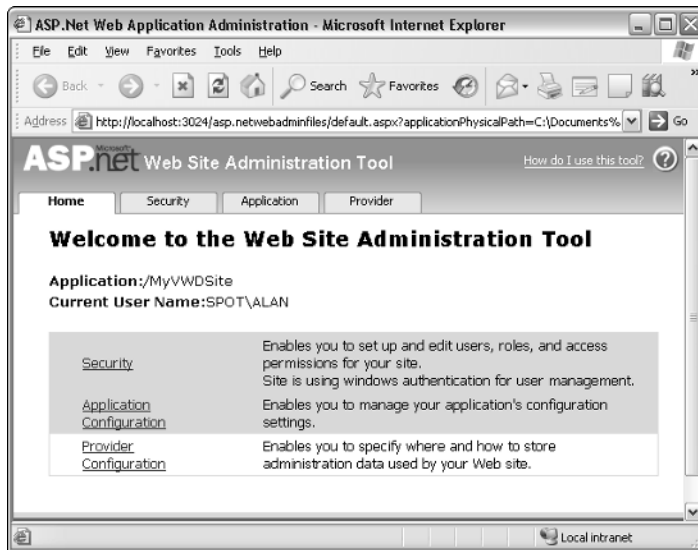


Figure 3-2:
The Home
tab in the
Web Site
Administra-
tion Tool.

3. Make sure that **AspNetSqlProvider** is selected, as shown in Figure 3-3. (Most likely there won't be any other options to select anyway).

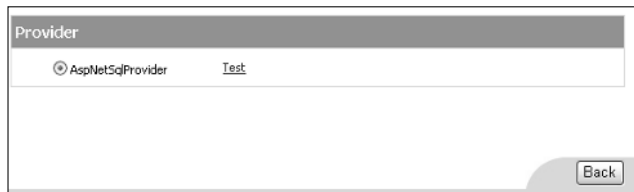


Figure 3-3:
Link to test
your SQL
Server
connection.

4. Click the **Test** link to the right of your selection to make sure your site can connect.
5. You should see a message that reads “Successfully established a connection to the database.” Click OK.
If the test fails the first time, just try again. Sometimes it takes a few tests for the connection to work.
6. Click the **Back** button in the lower-right corner of the page.
7. Click the **Home** tab in the Web Site Administration Tool.

At this point, you’ve told VWD that you want to use SQL Server as your site’s database management system for storing information about users and user accounts. The next step is to choose an authentication type.

Choosing an authentication type

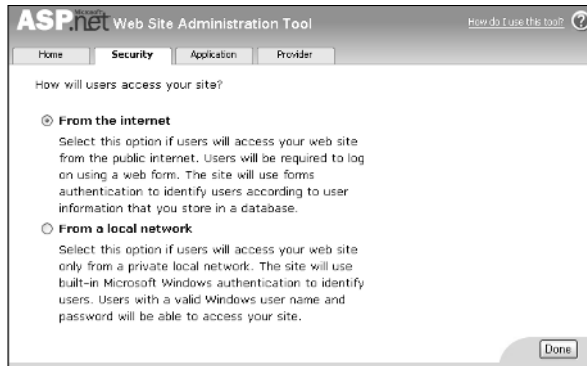
If you’ve ever set up any kind of an account on any Web site, you know that in order to log in you must enter two pieces of information: Your user name (or e-mail address) and your password. The user name defines who you are. Your password verifies that you actually are who you say you are (assuming you haven’t been handing out your password to people). That two-step process is known as *authentication* because it both identifies you and verifies that you are who you say you are.

In Visual Web Developer, you can choose between two forms of authentication: Windows authentication or Forms authentication. Windows authentication works only on a small Web site used within a local network. Assuming you intend on putting your site on the Internet, you’ll need to follow these steps to choose Forms authentication:

1. In the Web Site Administration Tool, click the **Security** tab.
2. Click the **Select Authentication Type** option under **Users** on the **Security** tab.
3. On the next page (Figure 3-4) choose the **From the Internet** option.
4. Click the **Done** button.

With the data provider and authentication mode selected, you’re ready to set up your site’s security by defining the access rules of your site. The first step is to create a role that will distinguish members of your site from anonymous users who are just looking around.

Figure 3-4:
Choosing
Forms
authenticat-
ion for an
Internet
Web site.



Creating Roles to Categorize People

If you think about how a business is organized, access to business resources and information tends to be based on people's *roles* within the company. For example, executives might have access to everything, managers have access to sensitive information within their department, and workers have access to whatever they need to do their jobs. Your role in the company determines your access to information.

In a membership Web site, access to information is also based on roles. A person who just browses to the site is an anonymous user because he hasn't signed into an account to identify who he is. To separate the anonymous users from people who are members with accounts, you need a role, perhaps named SiteMembers. In order for someone to be considered a SiteMember, she must have an account on the site, and must log into that account. After she's logged into her account, the person is no longer in the anonymous user role. Instead, that person is in the SiteMembers role.

To use roles in your Web site, you first have to enable that feature via the Web Site Administration Tool by following these steps:

- 1. If you haven't already done so, click the Security tab in the Web Site Administration Tool.**
- 2. If roles aren't already enabled, click the [Enable Roles](#) link.**

The center pane near the bottom of the Security tab now provides a link titled [Create or Manage Roles](#) as shown in Figure 3-5.



Do not create a role for anonymous users. Anyone visiting your site is by default an anonymous user, and you need not define such a role in the Web Site Administration Tool.

Figure 3-5:

Roles are enabled, so you can create or manage roles.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: 0 Create user Manage users Select authentication type	Existing roles: 0 Disable Roles Create or Manage roles	Create access rules Manage access rules

After you've enabled roles, you can define roles for people who visit your site. In a simple membership site, you really only need one role, like SiteMembers. To create a role, follow these steps:

1. **On the Security tab of the Web Site Administration Tool, click the Create or Manage Roles option.**
2. **Under the Create New Role heading on the next page, type a role name.**

For example, in Figure 3-6 I'm about to create a role named SiteMembers.

3. **Click the Add Role button.**

Figure 3-6:

About to create a role named SiteMembers.

You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.

Create New Role

New role name:

The page changes slightly to show the roles you've already created. For example, in Figure 3-7, you can see where I've created the SiteMembers role. You could, at this point, create more roles by entering role names and clicking Add Role for each role you want to create. For this example I'll just stick with the SiteMembers role. When you're done creating roles, click the Back button.

After you've clicked the Back button, you're returned to the Security tab of the Web Site Administration Tool.



Sometimes it helps to hear things explained in two or more ways. For a different explanation on using the Web Site Administration Tool, click the [How Do I Use This Tool?](#) link near the upper-right corner of the Web Site Administration Tool. Then click the Web Site Administration Tool – Security tab in the Overview page that opens. Also, pay attention to the text that appears on every page that opens in the Web Site Administration Tool.

Figure 3-7:
The Site-Members role is created and added to the list of role names.

Role Name	Add/Remove Users
SiteMembers	Manage Delete

Creating Access Rules

The whole purpose of creating roles is to distinguish among different types of people visiting your site, so you can control who has access to what. The way you do that is by defining *access rules* for different roles.

For example, at the start of this chapter I created a folder named MemberPages. The plan is to put all privileged content into that folder, to keep it separate from regular content that all users can view. But to make that happen, we need an access rule that *denies* anonymous users access to that MemberPages folder. You use the Web Site Administration tool to create access rules. Here are the steps:

1. In the Web Site Administration Tool, click the Security tab (if you aren't already there).
2. In the column titled Access Rules, click the [Create Access Rules](#) link.
3. In the left column of the resulting Add New Access Rule page, click the name of a content folder for which you want to create a rule.

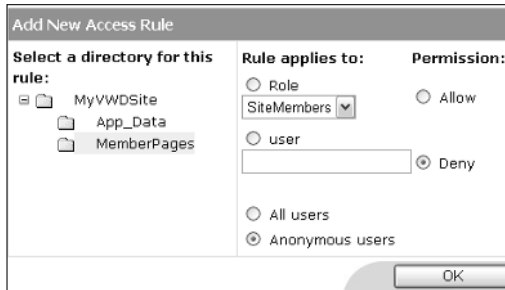


The term *content folder* means regular folders you created yourself, like my MemberPages folder. There's no need to assign permissions to the root folder at the top, or special folders like App_Data. Those predefined special folders already have all the permissions they need for the site to work correctly but remain secure. Changing those permissions will likely cause a world of confusion and problems for you. And all for naught because you shouldn't have messed with them in the first place.

4. In the center column, choose the role for which you want to create an access rule.

For example, in Figure 3-8, I've chosen MemberPages as the folder for which I want to create a rule. In the middle column I've chosen Anonymous Users as the role for which I want to create a rule.

Figure 3-8: SiteMembers is the selected folder; Anonymous Users is the selected role.



5. In the Permission column, choose whether you want to Allow or Deny people in the selected role access to the selected folder.

For example, in Figure 3-8, I've chosen the Deny option. In other words, the three selections made in Figure 3-8 define a rule that says "Anonymous users cannot access pages in the folder named MemberPages."

6. Click OK.

At this point, you have defined one access rule. There may be times when you need to manage (view, change, or delete) access rules you've already defined. Which brings us to . . .

Managing access rules

To review, change, or delete access rules you've defined in the past, click the [Manage Access Rules](#) link on the Security tab of the Web Site Administration Tool. The Manage Access Rules page of the tool opens.

Why does it say "Select a directory"?

You might have noticed the term "directory" under "Add New Access Rule" in Figure 3-8. The term *directory* is just another word for *folder*. *Directory* is the older term, harkening back to the early days of computing before there were icons on the screen to represent such things. The term *folder* is used more often

now, because icons that represent folders always look like little manila file folders. That's because a computer folder (or directory) is a "container" in which you store files, just as a folder in a file cabinet is a container in which you store paper documents (files).

It's important to keep in mind that access rules are defined on a folder-by-folder basis. Therefore, in the left column of the page that opens, your first task will be to click on the name of the folder for which you want to view or change access rules.

For example, in Figure 3-9 I've already clicked on Manage Access Rules to get to the page shown. I've also already clicked on MemberPages in the left column to see access rules for that specific folder. The center column shows all the rules defined (so far) for that MemberPages folder.

Figure 3-9:
Viewing
access
rules for the
Member-
Pages
folder.



Notice that with MemberPages selected in the left column, the first rule denies anonymous users access to the MemberPages folder. Below that is a dimmed, unchangeable rule that reads Allow [all]. It's important to understand that rules are applied in top-to-bottom order. Because the Deny permission is listed first, anonymous users won't be able to access pages in the MemberPages folder. However, everyone else *will* have access to that folder.

In this example, "everyone else" means SiteMembers, because that's the only other role a person could possibly belong to in this particular Web site because it's the only other role we've created.

There is no way, and no reason, for you to change, remove, or worry about that dimmed Allow rule that applies to [all] users. That rule is fixed and based on the most fundamental (yet unspoken) rule of all, which can be stated like this:

If you have information you don't want to share with others, don't publish that information on the Internet.

The Allow [all] rule is based on the simple fact that if you're publishing information on the Web, you're doing so because you *do* want to share that information with other people. The Deny access rule just *refines* that basic assumption by saying "I do want to share information in the MemberPages folder with *some* people who visit my site. I just don't want to share that information with anonymous users who visit the site."

That's all you need to prevent anonymous users from accessing members-only content in the MemberPages folder. So let's take a moment to review, in English, what you've done in the above steps:

- ✔ Created a folder named MemberPages for storing pages to be published on the Web.
- ✔ Created a role named SiteMembers that defines people who have an account and are logged into an account (as opposed to anonymous users).
- ✔ Created an access rule that prevents anonymous users from getting to members-only content in the MemberPages folder. SiteMembers will have access to that content (because there's no Deny rule preventing them from viewing the contents of that folder).

All of that is just an example. A site can have many content folders, many roles, and many access rules to control exactly who has access to what. But in the interest of keeping it simple and sane, we'll stick with the relatively easy example you've already defined here: The site contains a folder named MemberPages in which you can put members-only content.

At the moment, there's no way to test whether or not this will actually work. One reason why is that there is no user account on the site, and hence nobody in the SiteMembers role. For another, there is no content in the MemberPages folder to try to access. We can deal with the first problem by creating a hypothetical user account and putting that user in the SiteMembers role. As you'll see next, you can use the Web Site Administration Tool to create that user account.

Creating a User Account

You need at least one user account to work with while developing your Web site, just for testing and debugging purposes. You might as well create an account for yourself. To do so, you need to be in the Web Site Administration tool. If you've been following along, you're already in that tool. Otherwise, choose Website → ASP.NET Configuration to get into the tool.

In the Web Site Administration tool, click the Security tab. Then click Create User in the left column under the Users heading. Doing so will take you to the Create User page where you can define a user account.

Because the account is purely hypothetical, it doesn't much matter what you enter as a User Name, E-mail address, Security Question, and Security Answer. However, the password must be at least seven characters in length and must contain at least one non-alphanumeric character (that is, a punctuation mark). So, for testing purposes, use a password like:

```
password!
```

Active versus inactive users

Your membership system can consist of active and inactive accounts — which can be handy in a situation where people pay dues to maintain their accounts. If a member stops paying her dues, you can keep her in the database as an inactive user. You can then keep the user out

of privileged content, but still have the user's information in the database. This is better than deleting the user account because you can use the information in the database to send e-mail messages to inactive users, reminding them to pay their dues.



Chapter 7 covers ways that you can relax the password requirements so that users can enter passwords they're more likely to remember.

Because the real goal of this hypothetical account is to make sure the SiteMembers role works as intended, you need to select (check) the SiteMembers check box to the right of the Create User form. Figure 3-10 shows an example where I've created a hypothetical user named TestMember, and have placed that user in the SiteMembers role.

Create User	Roles
Sign Up for Your New Account	
User Name: <input type="text" value="TestMember"/>	Select roles for this user:
Password: <input type="password" value="••••••"/>	<input checked="" type="checkbox"/> SiteMembers
Confirm Password: <input type="password" value="••••••"/>	
E-mail: <input type="text" value="alan@coolnerds.com"/>	
Security Question: <input type="text" value="Favorite guitarist"/>	
Security Answer: <input type="text" value="hendrix"/>	
<input type="button" value="Create User"/>	
<input checked="" type="checkbox"/> Active User	

Figure 3-10:
A hypothetical user named TestMember.

Near the bottom of the form you'll notice an Active User check box. You'll want to make sure that box is selected (checked). Then just click the Create User button. You'll get some feedback indicating that the account has been successfully created. Click Continue to return to the Create User page. Because you only need one user account to test things out, you're finished here. Click the Back button in the lower-right corner of the Create User page to return to the Security tab of the Web Site Administration tool.

Managing user accounts

The [Manage Users](#) link on the Security tab provides a means of finding, editing, and deleting user accounts. Okay, when you have only a few user accounts to worry about, there's not much to “manage.” But as your Web site grows, so will the number of user accounts you manage (that's the idea, anyway).

Clicking the [Manage Users](#) link on the Security tab takes you to the page shown in Figure 3-11. There you can search for — and make changes to — user accounts.

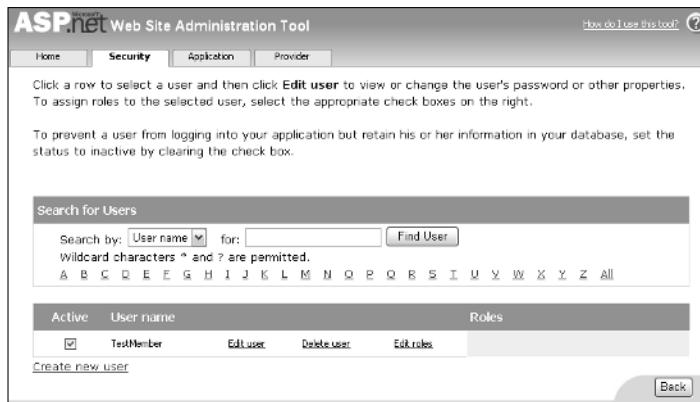


Figure 3-11:
Managing
user
accounts.

When you have lots of accounts to manage, use the options under “Search for Users” to locate any user account based on user name or e-mail address. As you’re typing your search text, you can use the ? and * wildcards as follows:

- ✓ ? Matches any single character
- ✓ * Matches any number of characters

For example, if you choose “E-mail” from the Search By drop-down list, enter ***@aol.com** as the text to search for, and click Find User, you’ll see all users whose e-mail addresses end in @aol.com.

Optionally, you can jump to any part of the alphabet by clicking any underlined letter. To see all user accounts, click the [All](#) link at the end of the alphabet. For example, if you perform a search that returns no results, you can click the [All](#) link and instantly re-display all user accounts.

After you find the account you want to change or delete, you can use the controls that are on the same row as the user name in the following manner:

- ✓ **Active check box:** Choosing the check box identifies the account as Active. Clearing the check box makes the account inactive (though does not remove the account from the database).
- ✓ **Edit user:** Clicking this link takes you to a page where you can change the account name, e-mail address, and role memberships of the user.
- ✓ **Delete user:** Deletes the user account. Unlike marking the account inactive, this option permanently removes the account from the database.
- ✓ **Edit roles:** Allows you to put the user into a role, or remove the user from a role.

To return to the Security tab options when you've finished managing user accounts, click the Back button in the lower-right corner of the page.

Closing the Web Site Administration tool

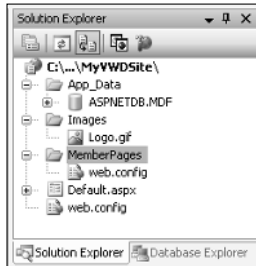
As with any other program, you can open, use, and close the Web Site Administration tool whenever you need to. Closing the Web browser that's showing the Web Site Administration tool automatically closes the tool; you'll be back to the Visual Web Developer program window.

What the Web Site Administration Tool Did

Back in Visual Web Developer's program window, all that work you did in the Web Site Administration tool appears as three new items in Solution Explorer. One of those items is a new database named `ASPNETDB.MDF`. To see its icon, click the + sign next to the `App_Data` folder in Solution Explorer, as shown in Figure 3-12. If you don't see that + sign, click the Refresh button in the Solution Explorer toolbar, or choose `View` ⇨ `Refresh` from the menu bar.

Down at the bottom of Solution Explorer is a new icon named `Web.config`. As its name implies, that file contains information about your site's general configuration. There's also a `Web.config` file in the `MemberPages` folder. That `Web.config` file contains code that prevents anonymous users from viewing pages in the `MemberPages` folder.

Figure 3-12:
ASPNETDB.
MDF and
Web.
config
icons in
Solution
Explorer.



There's no need for you to open either the `ASPNETDB.MDF` or `Web.config` files. They're all part of the site's infrastructure and don't relate to anything that appears on Web pages in your site.



A good general rule of thumb here is: If you don't know what something is, don't delete it, don't change it, don't rename it, and don't try to improve it by opening it up and hacking away at it cluelessly.

For now, we can consider the `ASPNETDB.MDF` and `Web.config` files “technical stuff” that is best left alone. The important thing to understand, though, is that the site now has all the infrastructure a site needs to support membership.

At the moment, there's no way to test that or to prove it. The site still needs a Login page so that you can log in and try things out. You'll create that page (`Login.aspx`) in Chapter 7. Furthermore, the `MemberPages` folder is still empty, so there's nothing to see in that folder.

In other words, you are not finished building your Web site. Not by a long shot. There's plenty more to be done. In Chapter 4, you'll discover how to create another important infrastructure component: a Master Page.

Chapter 4

Creating Master Pages

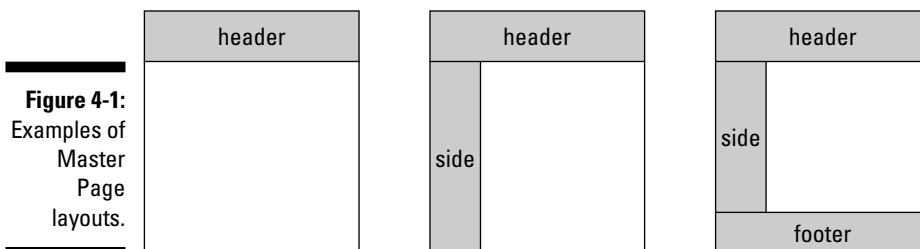
In This Chapter

- ▶ Giving your site a professional look and feel
- ▶ Creating a Master Page
- ▶ Using Master Pages
- ▶ Adding Master Page content to existing pages

A professional-quality Web site needs a consistent look and feel that lets users know they're still in your site as they move from page to page. The site must also be easy to navigate so users can get around without getting lost. *Master Pages* are a great way to give your site that consistent look and feel, because they allow you to define content that appears on every page in your site.

A Master Page lets you define a general format for all the pages in your site. For example, you might want a consistent header across the top of each page, to show your logo on each page. Or you might want a bar down the side of the page for displaying links and navigation controls. You might want a footer at the bottom of each page providing still more links. Or you may want a combination of header, sidebar, and footer.

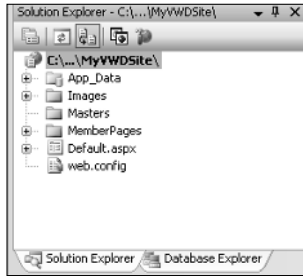
Figure 4-1 shows some general examples of layouts. The gray area on each page will be the same on every page in your site. The white area will be unique to each page. This chapter uses the middle example, in which you have a header and sidebar on each page.



Creating a Folder for Master Pages

You can put a Master Page wherever you like within your site. In the interest of staying organized, you may want to create a regular folder just for master pages. To do so, right-click the site path at the top of Solution Explorer and choose New. Then give the folder a name. I named my folder `Masters`, as shown in Figure 4-2.

Figure 4-2:
Adding a
new folder
named
`Masters`
to the site.



Creating a Master Page

Creating a new Master Page is similar to creating any other type of page in VWD. Here are the steps:

1. **In Solution Explorer, right-click the folder in which you want to place the Master Page and choose Add New Item.**

In my example I'd right-click my `Masters` folder.

2. **In the Add New Item dialog box, click Master Page.**
3. **Enter a name for your page, or just accept the default name `MasterPage.master`.**
4. **Choose a programming language.**

I use Visual C# in my own examples.

5. **Select the "Place code in separate file" check box.**

Figure 4-3 shows how the Add New Item dialog box might look at this point.

6. **Click the Add button.**

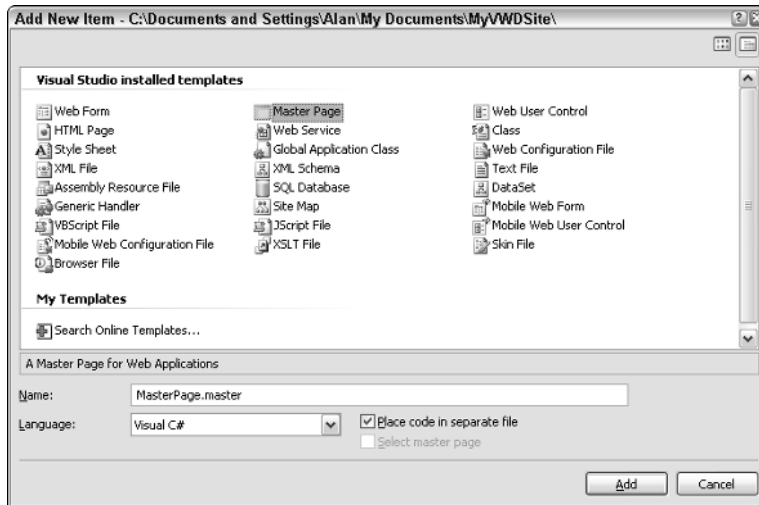


Figure 4-3:
Creating a
Master
Page.

A blank Master Page opens in Source view. If you click the Design button at the bottom of the Design surface, you'll see a page with a `ContentPlaceHolder` on it. That `ContentPlaceHolder` is the on-screen place where each page in your site appears.

Designing your page layout

Before you add anything to the Master Page, you want to choose your layout. Here's how:

- 1. From the VWD menu bar, choose `Layout` → `Insert Table`.**
- 2. In the Insert Table dialog box that opens, choose `Template`.**
- 3. Choose a template from the drop-down list.**

For example, in Figure 4-4, I've chosen the Header and Side layout.

- 4. Click `OK`.**

The Master Page is split into panes that reflect the options you choose. The `ContentPlaceHolder` will likely drop to the bottom of the page — perhaps out of view altogether until you scroll down through the page. In the design I chose, I wanted the content that's unique to each page to appear to the right of the side pane, not under the pane. So I needed to move the `ContentPlaceHolder` control into the appropriate cell on the page.

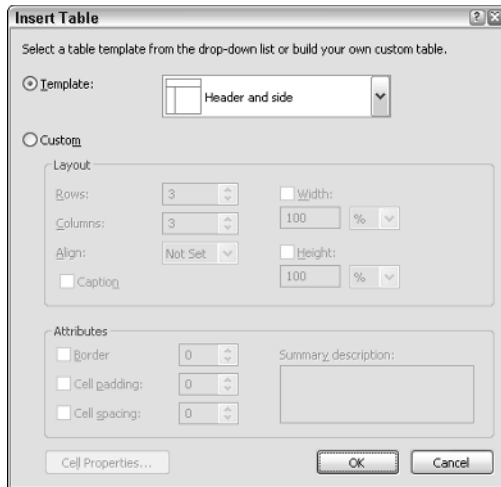


Figure 4-4:
Choosing a
page layout.



The page layout is really just an HTML table, and each “pane” is really just a table cell defined by a pair of HTML `<td>` and `</td>` tags. You can see that for yourself by clicking the Source button and viewing the HTML that defines the table.

To move an object, like the `ContentPlaceHolder` control, in Design view, you can either drag it, or cut and paste it. Here I just need to scroll down to, and click, the `ContentPlaceHolder` control to select it. Then just drag the four-headed arrow that appears into the bottom-right table cell and release the mouse button. Figure 4-5 shows what happens when you move the `ContentPlaceHolder` under the header pane and beside the left pane.

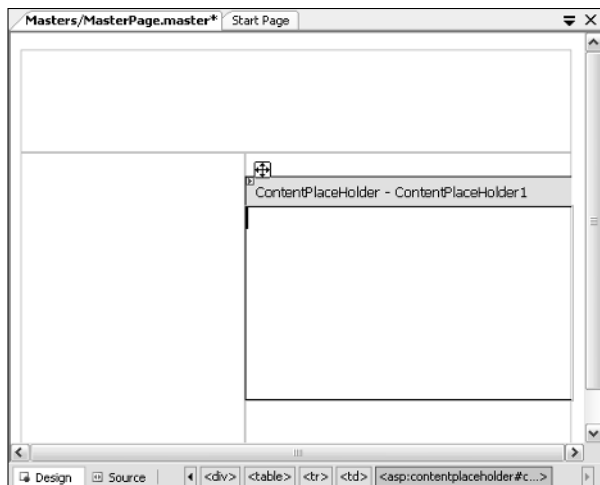


Figure 4-5:
Here's the
Content
Place
Holder
moved
into
the table.

Styling Master Page panes

Each pane in the Master Page is actually just a table cell — a very bland table cell in that each one has a white background and gray border. Even if you don't know exactly what you intend to put in each pane, you may want to size and color those panes and borders.

Any time you want to style something on a page, the Style Builder is your best bet. For one reason, it's any easy way to style things. For another, it follows Cascading Style Sheets (CSS) specifications, which is a good thing from a technical standpoint; these days, the trend is to use CSS to style everything in Web pages.



CSS is an Internet standard for styling elements on Web pages. Chapter 6 discusses CSS in some detail. But for the complete lowdown, see the official specification at www.w3.org/Style/CSS/.

To get to the Style Builder, right-click the item you want to style and choose Style. For instance, to style the top pane of the Master Page, right-click within that pane and choose Style. The Style Builder shown in Figure 4-6 opens.

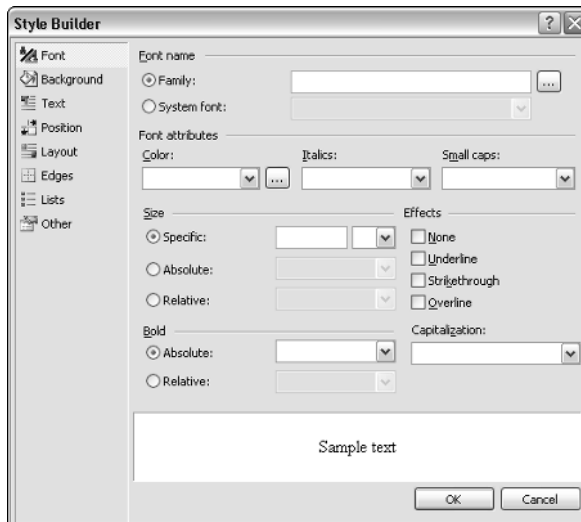


Figure 4-6:
The Style
Builder.

Choosing a background color

To choose a background color for the pane you're styling, click Background at the left side of the Style Builder, then choose a color from the Color

drop-down list. Or, click the Build button to the right of the Color drop-down list. Clicking the Build button opens the Color Picker dialog box, where you'll have more colors to choose from. You can choose any color from any tab just by clicking the color, and then clicking the OK button in the Color Picture dialog box.

When you click OK, the Color Picker dialog box closes. The Style Builder then shows the name of the color you chose or its HTML code (like #ffffffcc). The Sample box at the bottom of the Style Builder shows the selected color, as shown in Figure 4-7.

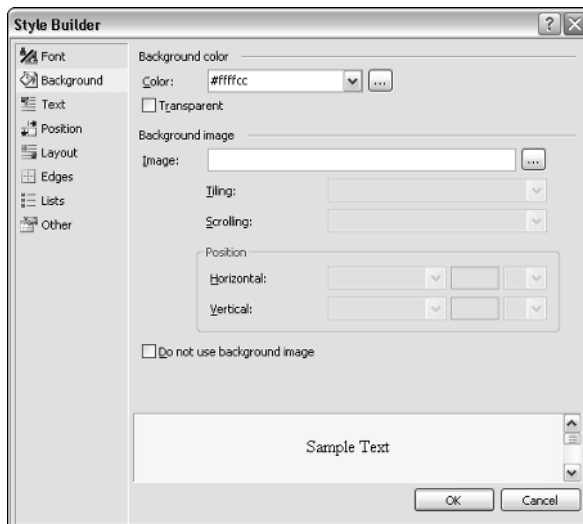


Figure 4-7:
Choosing a
background
color for
a cell.

Setting text alignment defaults

You can set a default horizontal and vertical alignment for text within the selected cell. At the left side of the Style Builder, click the Text option. Then use the Horizontal drop-down list to choose how you want text aligned within the cell.

For example, when styling the top cell you might want to choose Left as the Horizontal alignment and Bottom as the Vertical alignment as shown in Figure 4-8. Later, when you add text to the cell, that text will align to the left side of the cell and run along the bottom of the cell.

Setting cell height and width

To set the height or width of the selected cell, click Position at the left side of the Style Builder. Then you can use the Height and Width options to set the

cell's height and width. When you're styling the top pane of a Master Page, you'll likely want to set the width to 100%, because the pane needs to be as wide as the page. But you can set the width of the cell to any value you like. Figure 4-9 shows an example where I've set the height to 50 pixels and the width to 100%.

Figure 4-8:
Choosing
text
alignment
options for
the top cell.

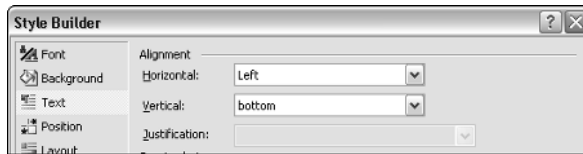
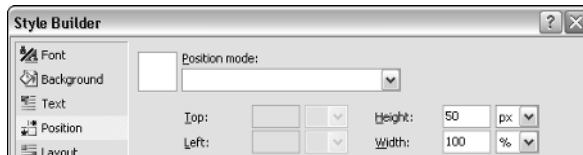


Figure 4-9:
Setting a
height and
width for the
top cell.

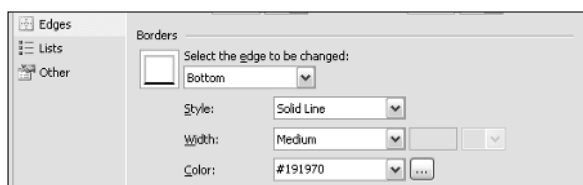


Styling cell borders

Every table cell has borders around it that you can color. You can style all the borders so they're the same or you can style borders individually. To get started, first click Edges at the left side of the Style Builder. Then, under Borders, choose which borders you want to style. In this example, when styling the top pane of a Master Page, you'd likely choose Bottom to style the line along the bottom of that pane.

Next, choose a style, width, and color from the respective drop-down list options. For example, if you want the line along the bottom of the top pane to be a little thicker and a little darker than the default gray line, choose Solid Line, Medium, and some dark color of your own choosing, as shown in Figure 4-10. (The color #191970 is just a dark blue I chose for the sake of example. Feel free to choose any color you like.)

Figure 4-10:
Styling the
top cell's
bottom
border.



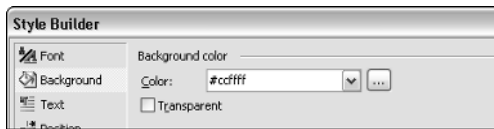
When you've finished making your selections in the Style Builder, click OK. The Style Builder closes and your selections are applied to the cell you styled.

Styling the left pane

The examples above work for the top pane of a Master Page. To style the left pane, right-click some empty space in that left pane and choose Style. Again, the Style Builder opens. But this time the selections you make are applied to the left pane only.

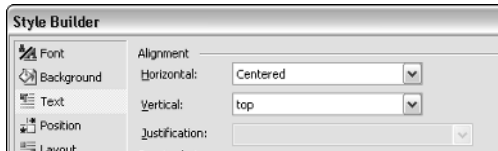
For example, maybe you want to change the background color in the left pane. To do so, click Background, then use the Color Builder to choose a different color. In Figure 4-11, I chose a light blue, which shows up as #ccffff.

Figure 4-11:
Choosing a light blue color for the left pane.



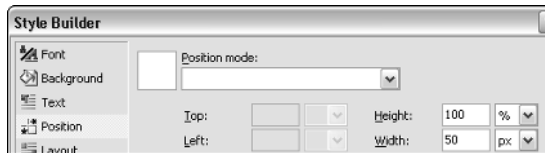
For text alignment in the left pane, click Text in the Style Builder. To center text in the left pane, and make it align toward the top of the pane, set the Horizontal and Vertical options to Centered and Top, respectively, as shown in Figure 4-12.

Figure 4-12:
Text alignment for the left pane.



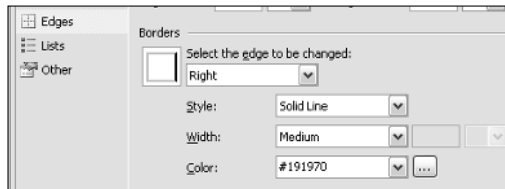
As a rule, you'll want the left pane's height to be equal to the height of the browser window. The width of that pane can be anything you like. To set the height and width, click Position in the Style Builder. Then set the height to 100% and the width to however many pixels you think you'll need. If you're unsure, just pick any number — such as 50 pixels, as shown in Figure 4-13.

Figure 4-13:
Setting the
left pane's
height and
width.



You might also want to style the right border of that left pane. To do so, click Edges in the Style Builder. Then choose Right as the edge to change. Choose a style, width, and color. Figure 4-14 shows an example where #191970 is again just a dark blue color I chose. You can use any color you like.

Figure 4-14:
Styling the
left pane's
right border.

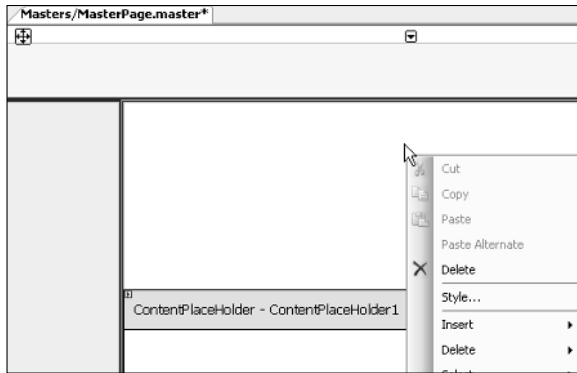


Click OK in the Style Builder to save your changes and apply them to the page. The left border then reflects the choices you made in the Style Builder.

Styling the ContentPlaceHolder pane

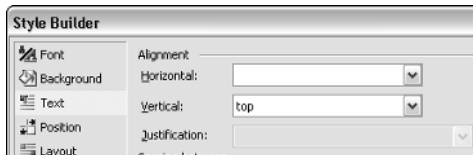
The pane that contains the `ContentPlaceHolder` doesn't really need to be styled in terms of color, width, or height, because the page that appears in place of the `ContentPlaceHolder` will eventually fill that pane completely. However, if you start with the `ContentPlaceHolder` aligned to the top of its table cell, you may find it easier to work with Master Pages. To make that change to the Master Page, first right-click some empty space within the `ContentPlaceHolder` pane. Make sure the mouse pointer is in the same cell as the `ContentPlaceHolder`, but not *on* the `ContentPlaceHolder` as shown in Figure 4-15. Then choose Style from the menu that appears.

Figure 4-15:
Right-click
the cell (but
not the
Content
Place
Holder
itself).



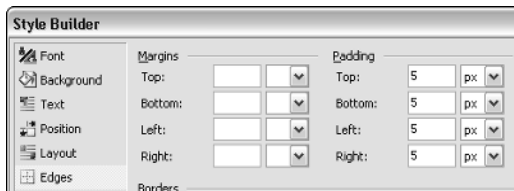
In the Style Builder, click on Text. Then choose Top as the Vertical alignment as shown in Figure 4-16.

Figure 4-16:
Alignment
setting
for the
Content
Place
Holder
pane.



You might find it useful to add some padding to the Content pane as well, because doing so provides a margin for content that will later appear in that pane. To pad the pane, click Edges in the Style Builder. Then set the Padding options to whatever you think is appropriate. For example, in Figure 4-17, I've set the padding inside that cell to 5 pixels all the way around.

Figure 4-17:
Padding the
Content
pane to
provide a
small
margin.



Click OK in the Style Builder to apply your style choices. The `ContentPlaceHolder` now aligns to the top of its cell. There's a small margin above and to the left of the `ContentPlaceHolder` because of the 5-pixel padding added in the Style Builder. The top and left panes have the color and border styles you applied, as shown in Figure 4-18. (Here in the book, of course, the colors I chose show up as shades of gray.)

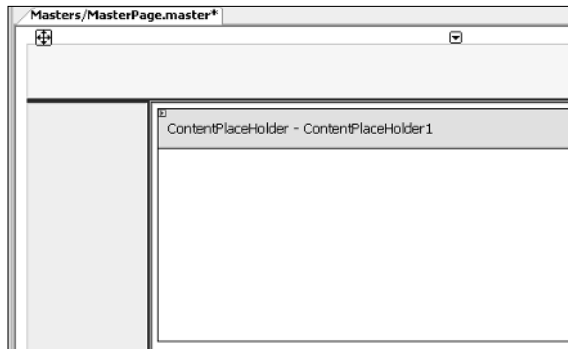


Figure 4-18:
The Master Page after a bit of styling.

Remember that Figure 4-18 is just an example: You can style things in your own Master Pages as you see fit. You haven't made any lifelong commitments here. If you change your mind about some stylistic setting you've made, just right-click the pane, choose *Style*, and use the Style Builder to choose whatever style options you like.

Eventually, you'll want to put some content (text, pictures, or whatever) into those panes. Keep in mind that whatever you put into those panes will be visible on every page that uses the Master Page. But, at this early stage of the design and development process, just having a Master Page with the basic look and feel you want is sufficient. You can worry about specific content later. Right now your time is better spent getting a feel for how you use Master Pages.

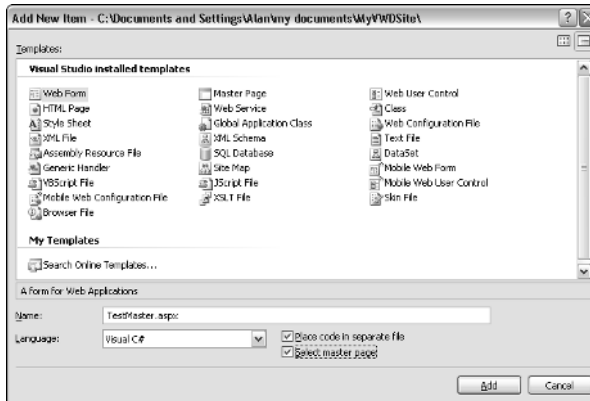
Before you can use a Master Page, you need to close and save it. Use the same technique you'd use to close any other item: Click the Close (X) button in the upper-right corner of the Design surface.

Using a Master Page

To use a Master Page, you must create a new Web Form. As you go through the process, choose the *Select Master Page* check box in the *Add New Item* dialog box. To try it out, create a simple page, perhaps named `TestMember.aspx`, by following these steps:

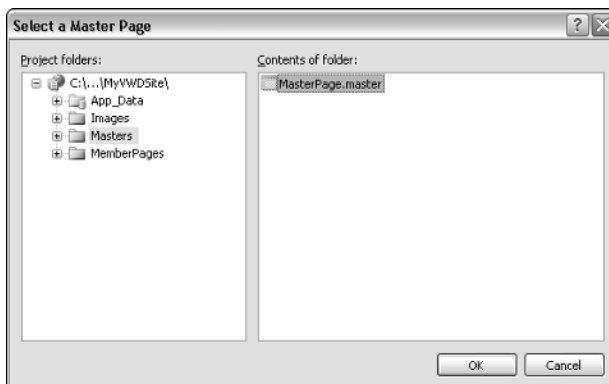
1. Right-click the site name at the top of Solution Explorer and choose Add New Item.
2. In the Add New Item window that opens, choose Web Form.
3. In the Name box, enter a name for the page.
4. I'll use the name `TestMaster.aspx` because this is just a test page.
5. Choose the Select Master Page check box as shown in Figure 4-19.

Figure 4-19:
Ready to create a new page that uses a Master Page.



6. Click the Add button.
7. In the Select a Master Page dialog box that opens, click the name of the folder that contains the Master Page (Masters, in this example). Then click the name of the Master Page you want to use, as shown in Figure 4-20.
8. Click OK.

Figure 4-20:
Choosing a Master Page to use.



Note that the page doesn't look exactly as it will in a Web browser; everything on the Master Page is dimmed and uneditable. That's because in this new page, you want to create and edit only the content of this one page, not the Master Page content, which appears on many pages.

The white area under the Content heading (which represents the `Content Placeholder` you see while creating the Master Page) is the only place you can compose your Web page. That white area is the *content page*, so named because it holds the content that's unique to the page you're creating right now. (As opposed to the Master Page, which shows content that appears on *all* pages that use the Master Page.)

Creating a content page is no different from creating a page that gives you the whole page to work with. For example, within the content page, you can type, edit, and format text using all the usual tools and techniques, as described in Chapter 2. You can add pictures by dragging their icons from Solution Explorer onto the page as was also described in Chapter 2. You can add controls from the Toolbox to the page. (Later chapters detail adding controls, and more basic formatting techniques as well.) Figure 4-21 shows an example in which I typed and formatted some text in the content page.

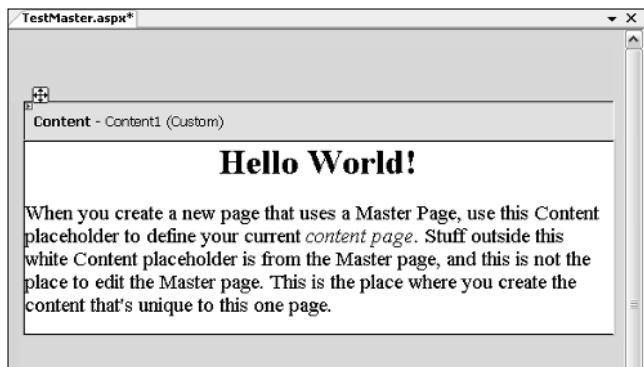


Figure 4-21:
New text in
the content
page.

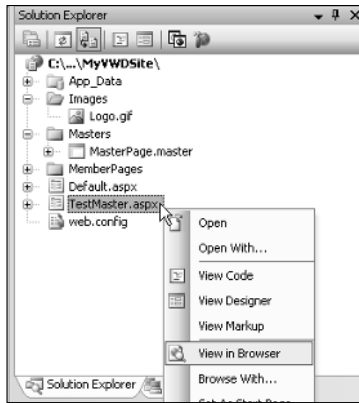


Don't worry about the small size of the content page in these figures, or on your own screen. Likewise, don't worry about the width of the content page's left pane. Things will adjust automatically as you work. It's nothing to be alarmed about because what you see in the Design surface isn't exactly how things will look in the Web browser.

When you've finished working on a content page, just close and save it as you would any other page. It will be placed in its folder with the usual `.aspx` extension. To get a better idea of how the page will actually look to people who visit your site, view the page in a Web browser. That is, right-click the page name (not the Master Page name) in Solution Explorer, as shown in Figure 4-22, and choose View in Browser.

Figure 4-22:

About to view TestMaster.aspx in a Web browser.



When the page opens in the browser, you'll see the panes from the Master Page above and to the left of the content page. At the moment, those panes are just strips of color (gray in Figure 4-23). But again, don't worry about the width of the panes or anything else. All that matters at this stage of the game is that you create a general layout for your Master Page, and understand how to use it when creating new pages. You can think about specific content later.

Figure 4-23:

TestMaster.aspx in a Web browser.



When you've finished viewing the TestMaster page, close your Web browser to return to Visual Web Developer.

Editing a Master Page

You can put anything you want into a Master Page. But it's important to realize that any time you want to edit the Master Page, you have to open the

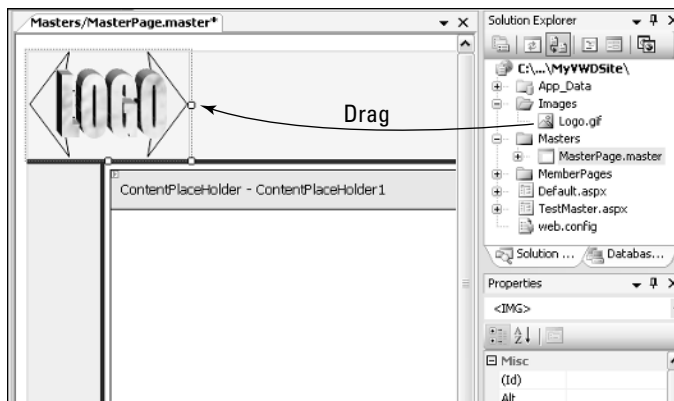
Master Page, not a page that just uses the Master. For example, to edit the Master Page created in this chapter, you'd double-click the `MasterPage.master` file, shown near the mouse pointer in Figure 4-24.

Figure 4-24:
To edit a Master Page, double-click its name in Solution Explorer.



When the page opens, you can add content to either the top or left pane as you see fit. The content can be anything — text, pictures, tables, controls, whatever. For example, to add a picture to the top-left pane of a Master Page, just drag a picture icon from Solution Explorer into the pane. Figure 4-25 shows an example where I've dragged a picture named `Logo.gif` to the top page of my Master Page.

Figure 4-25:
Adding a picture to a Master Page.



To add text to a pane, just click in the pane and type your text. After you've typed the text, you can select it and apply formatting as described back in Chapter 2. Figure 4-26 shows an example where I've typed the word `Welcome` in the left pane. I then selected that text and chose the Heading 1 style from the Block Format drop-down list.

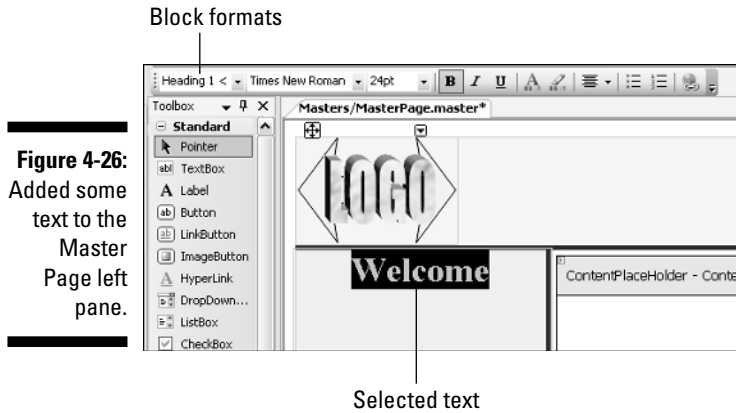


Figure 4-26:
Added some
text to the
Master
Page left
pane.

You can also add any control from the Toolbox to a Master Page. (Controls in the Toolbox are discussed in Chapter 7.) For example, if you want people to be able to log in from any page in your site, you could add a Login control to the pane. It's just a simple matter of dragging the control from the Toolbox into the top or left pane of the Master Page, as illustrated in Figure 4-27.



Figure 4-27:
Adding a
Login
control to
the left
pane.

As you add content to the left pane of the Master Page, the width of the pane changes to accommodate the content. Try not to let that bother you because it's not always a direct reflection of how things will look in a Web browser.

After making your changes to the Web page, close and save it. Then, open any page that uses the Master. For example, after making the sample changes in the previous three figures and closing the Master Page, I right-clicked my `TestMaster.aspx` page and choose View in Browser. Figure 4-28 shows the result.

Figure 4-28:
Test
Master.
aspx after
adding
content to
Master
Page.
master.



I'll be the first person to admit that the page in Figure 4-27 is ugly. But I'm not suggesting you create your Master Page to look like the one in the figure. All that matters now is that you *have* a Master Page that you can use as you create new pages in your site. You can think about what specific things you'd like to put on your own Master Page later.



Visual Web Developer is mostly about creating the infrastructure for a dynamic, data-driven Web site. You can worry about making things “pretty” later in the process. Figure 4-28 is nothing more than an example of how you *can* put things into a Master Page.

At this point, you do have a working Master Page, and you'll see more examples of ways you can use it in upcoming chapters.

Adding a Master Page to Existing Pages

If you've already created some pages in Visual Web Developer without specifying a Master Page, you can still get that content into a Master Page's Content area using simple copy-and-paste techniques. Actually, you already have a page that has no Master Page: the `Default.aspx` page that Visual Web Developer created automatically when you first created the Web site.

If you want the new page you're about to create to have the same name as the original page, you'll need to rename that original page first. To do so, right-click its name in Solution Explorer and choose Rename. Type a new name (without changing the `.aspx` extension) and press Enter. For example, you might rename `Default.aspx` to `OriginalDefault.aspx`.

Then, to copy and paste the stand-alone page's content into a new page's Content placeholder, follow these steps:

1. Double-click the stand-alone page to open it.

In my current example, that would be the `OriginalDefault.aspx` page.

2. If the page opens in Source view, click the Design button to switch to Design view.

3. Select and copy everything in the page (press Ctrl+A, then press Ctrl+C).

4. Right-click the site name at the top of Solution Explorer and choose Add New Item.

5. In the Add New Item dialog box, choose Web Form, and make sure you also choose (check) the Select Master Page check box.

6. Name the page whatever you like.

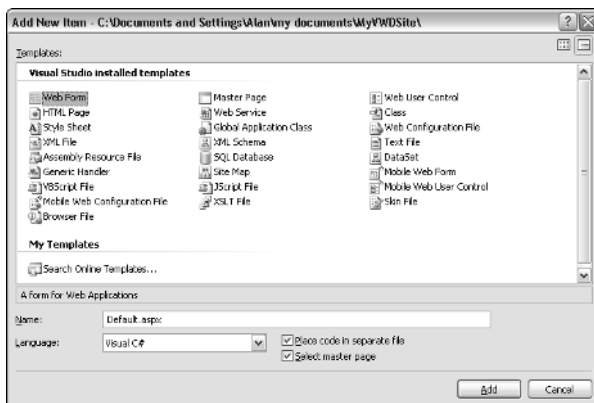
Figure 4-29 shows an example where I'm about to create a new `Default.aspx` page.

7. Click the Add button.

8. In the Select a Master Page box, click the folder that contains the Master, then click the Master Page.

For this example you'd click Masters in the left column, then click `MasterPage.master` in the right column.

Figure 4-29:
About to
create a
new
`Default.aspx`
page
with a
Master.



9. Click OK.

10. In the new page that opens, right-click the empty Content placeholder and choose Paste.

The copied content now appears in the Content placeholder. Now close and save the new page. To verify that things worked out as intended, view that page in a Web browser. Here, for instance, you'd right-click the new `Default.aspx` page in Solution Explorer and choose View in Browser.

The page opens in the browser, showing both the Master Page and the content you copied into its Content placeholder. Close the browser after checking things out.

Assuming the new page works as intended, you don't need the original page anymore because its content is in the Content placeholder of the new page you created. To delete the original page (named `OriginalDefault.aspx` in this example), right-click its name in Solution Explorer and choose Delete. Then choose Yes when asked for confirmation.

The next chapter starts Part II of this book, which gets into the nitty-gritty details of creating individual pages that take advantage of the infrastructure elements you've defined.

If pictures don't show in a Master Page

When you choose View in Browser to view a page that uses a Master page, pictures in the Master page should show up in the browser. If you see a red X where the picture should be, you need to make a slight change to the Master page. Here's how:

1. Close the browser, then double-click the Master Page in Solution Explorer to open it.
2. With the Master Page open for editing, click the Source button at the bottom of the design surface to switch to Source view. Locate the `<img...>` tag for the picture, which should look something like this:

```

```

3. Add `runat="server"` to the `` tag. Make sure there's a blank space before and after those new words and that the whole thing is inside the `<` and `>` tags as follows:

```

```

4. Close and save the Master page. Then view in a browser any page that uses the Master to verify that the picture works correctly.

Part II

Building Your Web Site

The 5th Wave By Rich Tennant



"Look into my Web site, Ms. Carruthers.
Look deep into its rotating, nicely
animated spiral, spinning, spinning, pulling
you in, deeper... deeper..."

In this part . . .

After you've laid out the basic foundation of your data-driven Web site, it's time to start creating pages. As you discover in this part, VWD pages use the same basic tools as any other Web page to format your site's content: HTML and CSS. Visual Web Developer also offers powerful ASP.NET controls that can do a lot more than HTML and CSS alone. In this part, you get a look at how all that fits together, and see an example of creating a control that makes your site easy for your visitors to navigate.

Chapter 5

Creating Web Pages

In This Chapter

- ▶ Creating and editing tables
 - ▶ Adding hyperlinks to pages
 - ▶ Adding and styling pictures
 - ▶ Working in Source view
-

No matter what program you use to create Web pages, you're actually creating a document that contains text and HTML tags. That's because all Web pages use HTML as a markup language.

This chapter builds on Part I of this book, which focused on building the foundation of your Web site. With the foundation done, you can start creating content for individual Web pages.

As you'll discover in this chapter, creating and editing Web pages in VWD is much like creating and editing word-processing documents — although some of the procedures differ from typing content into a program such as Microsoft Word or FrontPage. This chapter shows you what to look for, and covers all the basics of creating and editing pages in Visual Web Developer.

Creating a New Blank Page

Visual Web Developer offers you two types of Web pages to create:

- ✓ **Web Forms:** Can contain HTML and ASP.NET Server controls. The filename extension is `.aspx`.
- ✓ **HTML Pages:** Can contain HTML but no ASP.NET controls. The filename extension is `.html`.

So if you want your page to use any ASP.NET capabilities at all, even if you just want to use a Master Page with the page you're creating, you want to create a Web form. You would only use HTML to create pages that don't use a master, and don't use ASP.NET controls.

To create a new page, start in Solution Explorer. If you want to put the page in the root folder, right-click the site folder at the top of the Solution Explorer page and choose Add New Item. To put the page into a particular folder in Solution Explorer, right-click that folder's icon in Solution Explorer and choose Add New Item. Either way, the Add New Item dialog box opens.

If you want to create an HTML page that doesn't use ASP.NET server controls, click the HTML Page icon. The programming-related controls, and the option to select a Master Page, are dimmed because you can't use those items in an HTML page. Just type the filename for your new page and click the Add button. The page opens in the Design surface ready for editing in either Design or Source view.

To create a new `.aspx` page from the Add New Item dialog box, click Web Form; then choose your programming language, whether or not you want to put code in a separate file (you should always choose this option), and whether you want to include a Master Page. Then click the Add button. In Figure 5-1, for example, I'm about to create a new page named `Login.aspx` that uses C# as its programming language, places code behind the form, and uses a Master Page.

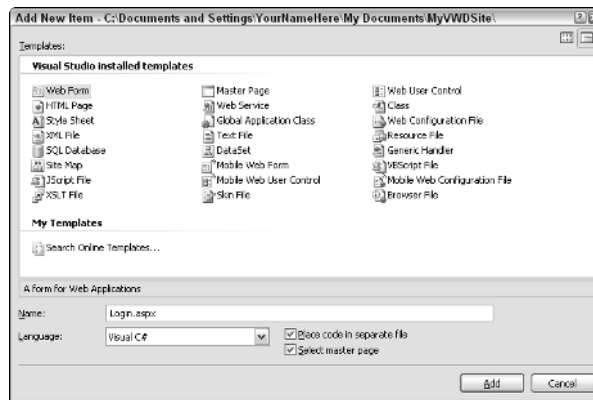


Figure 5-1:
Creating a
new `.aspx`
page (Web
form).

If you choose “Select master page,” then clicking the Add button calls up a dialog box that asks which Master Page you want to use. Just click the name of the folder that contains the Master Page, click that page's name, and then click OK.

If you opted to use a Master Page, you'll do all your work in the white Content box that appears on the page. Although the box doesn't look as large as a whole page, it grows to accommodate whatever you add. What's most important to

understand is that the techniques described in this chapter work the same whether you're creating an HTML page, a particular `.aspx` page without a master, or another `.aspx` page controlled by a Master Page.



After you have a page (whether `.aspx` or `.html`) open in the Design surface, use the Design and Source buttons at the bottom of the Design surface to switch between the WYSIWYG Design view and HTML Source view.

Creating HTML Tables

After you have a page open in Design view, creating the page is not too different from creating a document in a word processing program. Text flows as it would when typing a normal business letter. If you want to organize text into columns and rows, you can insert a table into the page and use its cells to organize content.

Adding a table to a page

You can add a table to any page (including the Content page in a page that uses a Master Page). One way to do so is as follows:

- 1. In your page, click where you want to put the table.**
- 2. Choose Layout → Insert Table from the menu bar.**
- 3. Choose the Custom option.**
- 4. Specify the number of rows and columns you want.**

For example, in Figure 5-2 I'm about to create a table with two rows and two columns.

- 5. Optionally, choose other formatting options under Layout and Attributes.**

Instead of choosing options in the Insert Table dialog box to format every table you create, you can use CSS to create a general style that applies to *all* tables. (Chapter 6 gives you the goods on CSS.) Here, it is sufficient to choose only the number of rows and columns you want.

- 6. Click OK.**

An empty table with the number of rows and columns you specified appears on the page.

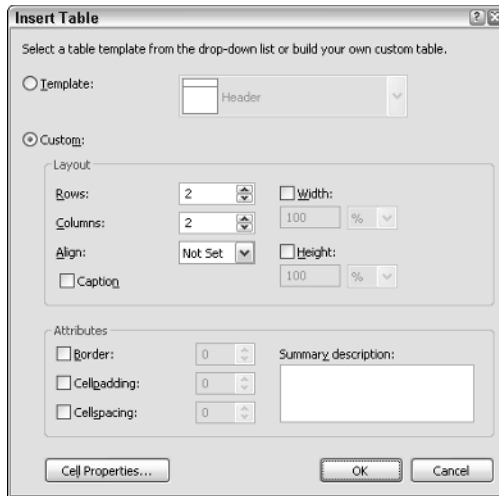


Figure 5-2:
Insert Table
dialog box.

Typing in table cells

If you have any experience at all with Microsoft Word or an HTML editor, you'll find there's nothing unique to working with text in table cells. To type in a cell, you just click the cell and start typing; Figure 5-3 shows where I've inserted a table with two rows and two columns. Then I typed a question into each of the top two cells. I widened each column by dragging its right border to the right until the text in the table cell no longer wrapped to two lines.

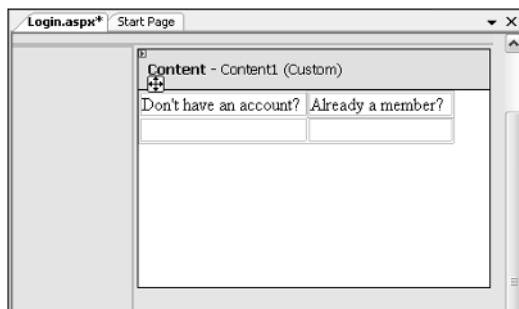


Figure 5-3:
Text typed
into top two
cells.

Working with HTML Tables

You can change a table or its contents at any time. Options for managing the table as a whole, and for inserting and deleting columns, are on the Layout menu in the toolbar — but make sure you click somewhere inside the table

before you choose options from that menu. Here's a quick rundown of how to perform common tasks:

- ✓ **Add a column:** Click a column that will be next to the new column, and then choose Layout⇨Insert⇨Columns to the Left or Columns to the Right.
- ✓ **Add a row:** Click a row that will be next to the new row, and then choose Layout⇨Insert⇨Rows Above or Rows Below.
- ✓ **Insert a cell:** Click the cell that's to the left or right of where you want the new cell to be placed. Then choose Layout⇨Insert⇨Cells to the Left or Cells to the Right.
- ✓ **Delete an entire table:** Click anywhere inside the table you've decided to delete. Then choose Layout⇨Delete⇨Table.
- ✓ **Delete a column:** Click in the column you want to remove and choose Layout⇨Delete⇨Column.
- ✓ **Delete a row:** Click in the row you want to remove and choose Layout⇨Delete⇨Row.
- ✓ **Delete a cell:** Click the cell you want to remove and choose Layout⇨Delete⇨Cell.
- ✓ **Resize a column:** Drag the right border of the column left or right. Or, click in the column, choose Layout⇨Resize⇨Resize Column, set your width, and then click OK.
- ✓ **Resize a row:** To make a row taller or shorter, drag its bottom border up or down. Or click the row, choose Layout⇨Resize⇨Resize Row, set the row height, and then click OK.



As always, if you don't like a change you've made to a table, press Ctrl+Z or choose Edit⇨Undo to undo that change.

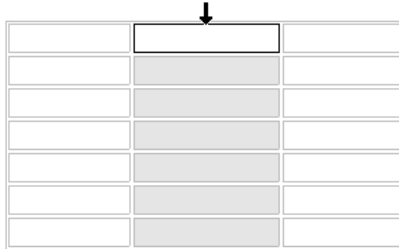
Selecting rows and columns

You can change the appearance of any cell, or its contents, at any time. You can work with individual cells. Or, you can select multiple cells first, and then apply your change to all of the selected cells.

There are several methods you can use. To select a column or row, first click in the column or row you want to select. Then right-click the cell and choose Select⇨Column or Select⇨Row. As an alternative to right-clicking, choose Format⇨Select⇨Column or Format⇨Select⇨Row.

You can also select a column by clicking its top border. Likewise, you can select a row by clicking its leftmost border. You have to get the tip of the mouse pointer right on the border, so the mouse pointer turns to a little black arrow first, like the example in Figure 5-4. Don't hold down the mouse button until you see the little black arrow.

Figure 5-4:
Little black
mouse
pointer for
selecting
table
columns.



To select multiple columns, you can drag the little black mouse pointer left or right through the top border of other columns. Likewise, drag the little black arrow up and down to select multiple rows in the table.



When you select multiple cells, one of the selected cells won't be gray. Instead it will be white with a black border like the top cell in Figure 5-4. That's normal. Don't drive yourself nuts trying to make the last selected cell gray. It's not necessary.

Selecting cells

To select a single cell, just click in the cell. To select multiple adjacent cells, drag the mouse pointer through the cells you want to select. To select multiple, nonadjacent cells, click in the first cell you want to select, and then hold down the Ctrl key as you click other cells. To deselect a single selected cell, Ctrl+Click that one selected cell.

Most of the selected cells will be highlighted (grayed). But as when selecting rows and columns, one selected cell will be white with a black border. When using the Ctrl+Click method, the last cell you select will be the white one.

Merging cells

You can merge two or more cells to form a single large cell. For example, if you want to center a heading in the first row of the table, you can merge all of its cells so it's just one long cell across the table. Then you can type any text you like in that cell and center it above the table.

To merge cells, select the cells you want to combine into a single cell. Then right-click any selected cell and choose Merge Cells. For example, the top of Figure 5-5 shows three cells across a table selected. Just below that is the result of merging those three cells.

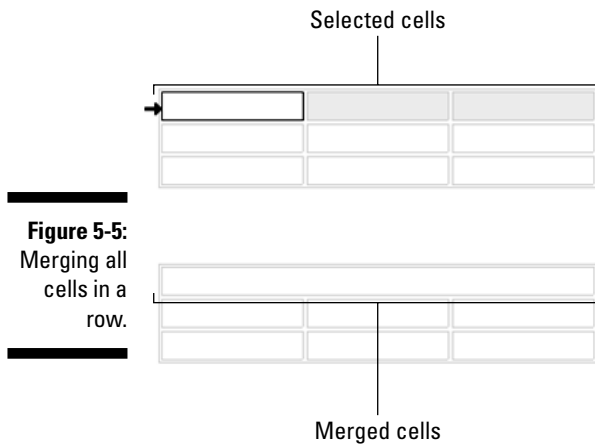


Figure 5-5:
Merging all
cells in a
row.

Styling cells

When it comes to styling things in a Web site, there are lots of ways to go. But because the world seems to be gravitating toward a universal XHTML standard, the Style Builder will be your best bet. First you have to make sure you know what you're about to style. Three rules apply:

- ✓ To style a single cell, click that cell (the Properties sheet will show `<TD>`, indicating that you're about to style a table cell).
- ✓ To style multiple cells, rows, or columns, select whatever you want to style. The Properties sheet won't show any tag when multiple items are selected.
- ✓ To select the entire table, right-click a cell and choose `Select↔Table`. The Properties sheet shows a `<TABLE>` tag, indicating that you're about to style the entire table.

Then you can get to the Style Builder by using either of these methods:

- ✓ Right-click the item or selection and choose Style. (If Style is disabled on the menu, use the next method instead.)
- ✓ Scroll down to, and click on, the `style` property in the Properties sheet, and then click the Build button that appears.

Either way, the Style Builder opens. Items down the left column represent different things you can style, such as the font, background, text, position, and so forth. When you click a category name, the main pane to the right shows options in that category.

For example, suppose you selected the top row across a table. In the Font category of the Style Builder, you can choose a font family and style for text within those cells. Figure 5-6 shows the font set to Arial, its Color to Navy, its size to 12 points, and its weight to Boldface. The Sample Text at the bottom of the screen shows you how things will look when your selected options are applied.

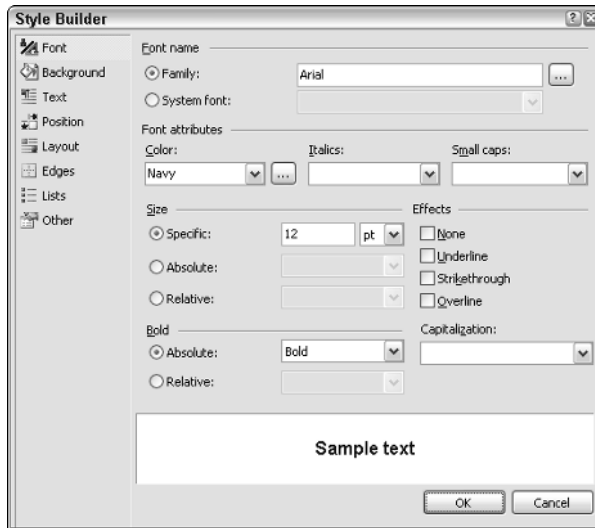


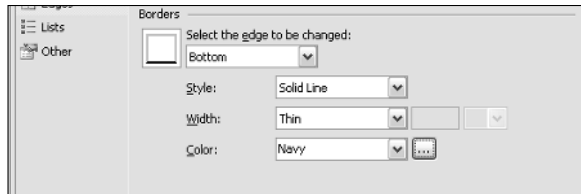
Figure 5-6:
Some Font
options
selected
in Style
Builder.

To add a little background color to the selected cells, click Background in the Style Builder. Then click the Build (...) button. In the Color Picker that opens, click any color you like, and then click OK. The preview sample at the bottom of the style builder shows how your text will look against the background color you chose.

To center text in all of the selected cells, click the Text category at the left side of the Style Builder, and then choose Centered from the Horizontal option under Alignment. To determine how text aligns vertically in a tall cell, choose either Top, Middle, or Bottom from the Vertical option under the Alignment heading.

To style the borders around the selected cell(s), click the Edges button. Under Borders, select which borders you want to style. In Figure 5-7, for example, I opted to style only the Bottom border as a thin, navy-blue solid line.

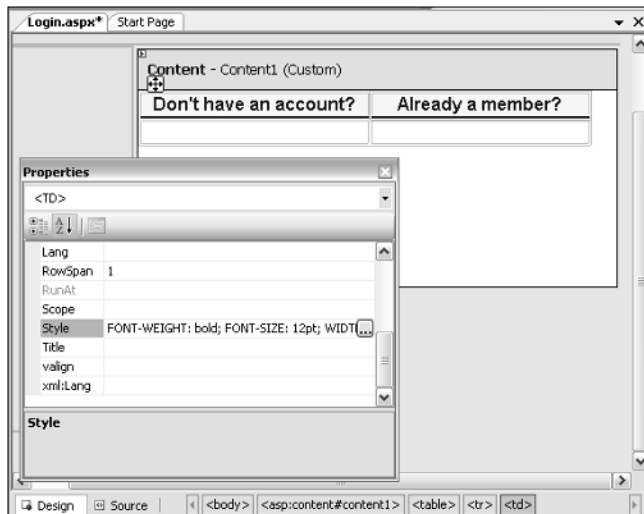
Figure 5-7:
Styling table
cell borders.



When you've finished making your selections in the Style Builder, click OK. The items you styled take on the chosen new style. Figure 5-8 shows an example where the top two cells now have a background color and dark border along the bottom. The text in each cell is 12pt Arial Bold, and centered within each cell.

Figure 5-8 also shows how the `style` property looks after clicking OK in the Style Builder. Your selections are converted to CSS formatting syntax. You can change the style at any time by clicking the `style` property again, and the Build button opens again, and you can make whatever changes you want.

Figure 5-8:
Two styled
table cells
and a
Properties
sheet.



If you just want to format a small chunk of text within a cell, select that small chunk of text only. Then use the Formatting toolbar to apply a font, color, italics, or whatever.

Chapter 6 will get more into the Style Builder. For now, that should be enough to get you working with tables in the VWD Design surface. But there's one other thing I should mention about tables in this chapter. And that is . . .

Adding controls to table cells

If you're creating an .aspx page, you can also add ASP.NET Web server controls to table cells. Doing so couldn't be easier: You just drag the control from the Toolbox into a table cell. In Figure 5-9, for example, I dragged a Login control from the Toolbox into the table cell under the heading "Already a member?"

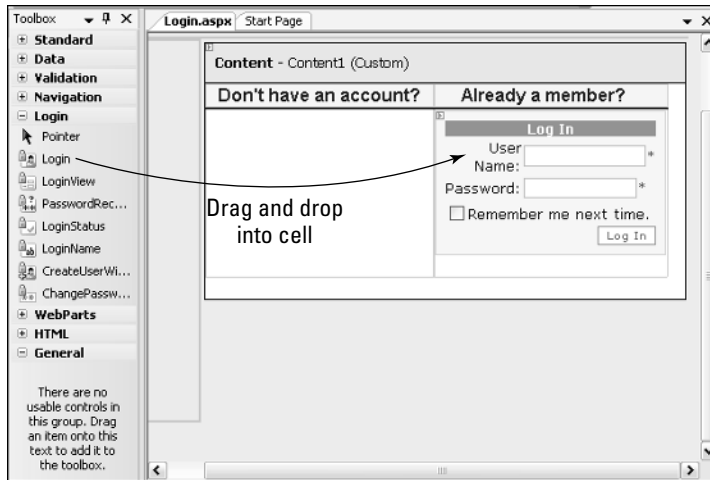


Figure 5-9:
Login
control
added to
a table.



Chapter 7 gives you more info on using and formatting ASP.NET controls like Login.

So like I said, that's the basics of working with tables. Now, on to another common design element found in Web pages.

Adding Hyperlinks to Pages

There are many ways to add hyperlinks to pages in VWD. Here's a method for creating a link to any page on the Web:

1. If you haven't already done so, type the text that you want to serve as the link.
2. Select the text that will act as a link.

In Figure 5-10, I've typed and selected the phrase "Sign up now!"

Figure 5-10:
Selected
text and
Convert to
Hyperlink
button.



3. Click the Convert to Hyperlink button in the Formatting toolbar.

In Figure 5-10, the mouse pointer is resting on the Convert to Hyperlink button. When you click this button, the Hyperlink dialog box appears.

4. Choose the link's Type.

For example, choose http: for a Web page.

5. Type (or paste) the complete URL of the target Web page.

6. Click OK.

The selected text shows as blue and underlined.



The blue-and-underlined text doesn't act as a link in VWD. To test the link, open the page in a Web browser.

Quick links to pages in your site

If you want to create a link to another page in your Web site, you can just drag that page's name from Solution Explorer onto the page. Initially, the text on the page will exactly match the filename of the page, as given here:

Default.aspx

To change the text of the link to something more meaningful than the file name, select the hyperlink text. Type your new text in its place. Changing the text of the link won't change the file to which the link refers. When you click the link to select it in your page, the Properties sheet will show an <A> tag (because all links are <A> tags in HTML).

Figure 5-11 shows an example where the cursor is in a link (Home). The Properties sheet shows an `<A>` tag, and the `HRef` property is the page that the link will open (when used in a Web browser).

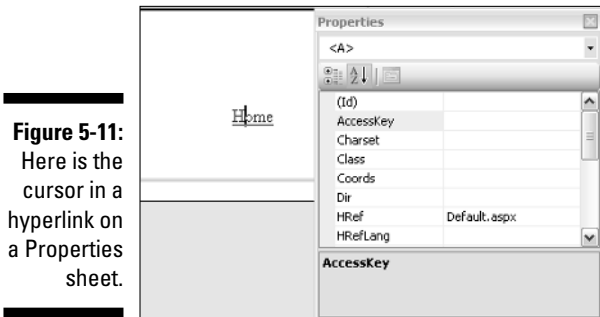


Figure 5-11:
Here is the cursor in a hyperlink on a Properties sheet.

Creating bookmarks

Your page can contain *bookmarks*, places that you can jump directly to from any link in the same page, or any link in any other page. To create a bookmark, click where you want to place the bookmark in your page, so the cursor is positioned where you want the bookmark to be. Then choose **Format**→**Insert Bookmark** from the menu bar. Type in a name for your bookmark (for example, **Top** for a bookmark at the top of a page). Then click **OK**.

You won't see any trace of the bookmark in Design view because bookmarks are not visible there (or in Web browsers). You can see the bookmark only in Source view, where it will be expressed in HTML and look something like this:

```
<a name=" YourName" ></a>
```

where *YourName* refers to whatever name you gave your bookmark.

Linking to bookmarks

To create a link to a bookmark from within the same page, follow these steps:

1. **Type the text of the link exactly as you would for creating a link to an external page.**

For more about this process, refer to the earlier section, "Adding Hyperlinks to Pages."

2. Select that text and click the Convert to Hyperlink toolbar button.
3. In the dialog box that opens, choose Other as the link type.

For the URL, type the bookmark name preceded by a pound sign (#). For instance, if you named a bookmark Top, you'd type `#Top` as the URL.

When you click the link in Source view, the Properties sheet shows an `<A>` tag. The `HRef` property is the one whose name has a leading pound sign (for example, `#Top`). As with any link, you can't really test these out in the VWD Design view. You have to open the page in a browser to test your links.

Adding and Styling Pictures

The easiest way to include pictures in your Web site is to first get them into a folder in the VWD Solution Explorer. As discussed in Chapter 2, you can just drag the picture icons from any folder in Windows Explorer to any folder in Solution Explorer.

To add a figure to your page, just drag the picture's icon from Solution Explorer onto your page. In Design view, the figure looks much as it will in a Web browser, though when it's selected on the page it will show borders that won't appear in the Web browser.

To select a picture, click it. After it's selected, the picture appears framed with a border showing three dragging handles (as in Figure 5-12). Also, the Properties sheet displays an `` tag; the `Src` (source) property will be the path to the image file, as is also shown in the figure.

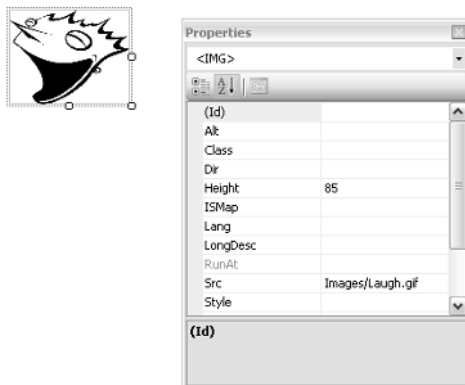


Figure 5-12:
Selected
picture and
Properties
sheet.

Sizing a picture

To size a picture interactively — changing its aspect ratio without distorting the picture — drag the handle in the lower-right corner of the selected picture. To widen or narrow the picture without regard to aspect ratio, drag either of the other two handles.

You can also set the height and the width, in pixels, using the Height and Width properties in the Properties sheet or the Style Builder. The Style Builder also offers options for controlling the position of the picture and its borders, so we'll look at those techniques next.

Styling pictures

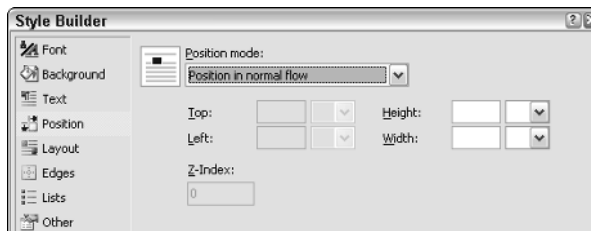
To style the picture, right-click it and choose Style, or, in the Properties sheet, click the `style` property (just under the `src` property), then click the Build button that appears. The Style Builder opens. Options relevant to pictures are in the Layout and Edges categories of the Style Builder, as the next subsections show.

Positioning pictures

When you click the Position category in the Style Builder (Figure 5-13), you can choose any of the following options from the Position Mode drop-down list to position the image:

- ✓ **Position in normal flow:** Anchors the figure to neighboring text so it sticks to the current paragraph. (This is the most common selection.)
- ✓ **Offset from normal flow:** Anchors the figure to the neighboring text, offset from where it would normally appear in the flow by whatever amount you specify in the Top and Left options.
- ✓ **Absolutely position:** Anchors the figure to a specific place on the page, independent of text flow.

Figure 5-13:
Position
options in
Style
Builder.



The Height and Width options let you set the height and width of the picture in pixels, points, percentage of the page width, or any other of several units of measure.

The Top and Left options are enabled only if you choose “Offset from normal flow” or “Absolutely position.” If you choose “Offset from normal flow,” you can define the offset using Top and Left. For example, entering 10px for Top and 10px for Left would position the picture 10 pixels down and to the right of where it would have been positioned if you chose “Position in normal flow.”

If you choose “Absolutely position” from the Position Mode drop-down list, the Top and Left options define where the picture is placed relative to the upper-left corner of the page. For example, setting each measurement to 10px would put the upper-left corner of the picture 10 pixels away from the upper-left corner of the page.

The Z-Index option applies only to absolutely positioned objects, and defines the layer in which the object exists. The page itself is always layer 0. Objects at layer 1 can cover the page. Objects at level 2 can cover objects at level 1, and so forth, just like sheets of paper with the bottom sheet being layer 0.



Note that anything at a lower layer will be covered by an object at a higher layer. So if you position a picture absolutely or offset it, there’s a good chance that the picture will cover the text beneath it. Not good.

Wrapping text around a picture

For text that’s positioned to flow with text (as opposed to positioning the images absolutely), click the Layout category name at the left side of the Style Builder. Then use the “Allow text to flow” option to set whether (and how) text flows around the figure. Your options, in a nutshell, are these:

- ✓ **To the right:** The picture aligns to the left margin and text flows to the right side of the picture, as in the top example in Figure 5-14.
- ✓ **Don’t allow text on sides:** No text flows around the picture, as in the example in the center of Figure 5-14.
- ✓ **To the left:** The picture aligns to the right margin and text flows down the left side of the picture, as in the bottom of Figure 5-14.

Bordering pictures

To put a border around a picture, or to increase the distance between a picture and the text that flows around it, use the Edges category in the Style Builder (Figure 5-15). In particular, you can use the Margins options to define the amount of space between the picture border and neighboring text.



Figure 5-14:
Text flow
and
pictures.

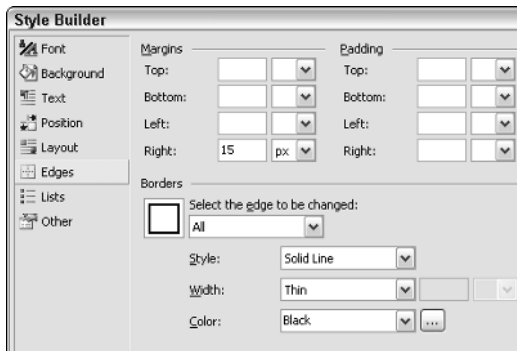


Figure 5-15:
The Edges
category in
Style
Builder.

If you want to put a border around your picture, choose All from the “Select the edge to be changed” drop-down list. Then choose the style, width, and color of the border. Click OK, and the selected picture will take on the new style.

Figure 5-16 shows an example that allows text to flow to the picture’s right. The border around the picture comes from the selections under Borders in Figure 5-15. The gap between the picture border and the text to the right of the picture is defined by the 15 pixel right margin setting in Figure 5-15.

Why can't I center a picture?

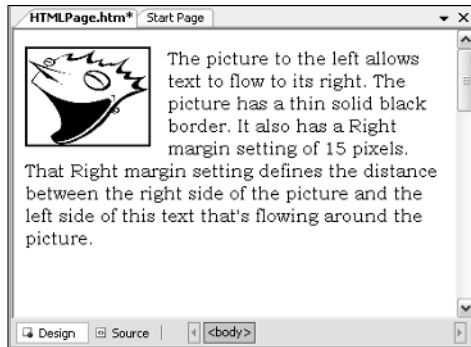
If you're familiar with HTML, you may be accustomed to using the `align` attribute in `` tags to align pictures. That attribute is being phased out in favor of using CSS styles to align pictures. To align a picture to the left or right margin, just choose the "To the left" or "To the right" option from the Allow Text To Flow drop-down list. To center a picture, enclose the `` tag in a pair of `<div>...</div>` tags that have the `text-align` property set to Center, as follows:

```
<div style="text-align:center">

</div>
```

You can type the required tags manually in Source view. (More on using the Source view a little later in this chapter.)

Figure 5-16:
Picture with
border and
right margin.



Padding pictures

The Padding options in the Edges category define the amount of empty space between the picture and its outer border. For example, Figure 5-16 shows the border actually touching the bottom of the picture (the character's chin). Increasing the Padding options to 5 pixels or more would increase the size of the box (but not the *picture* in the box) to put more room between the picture and the border surrounding the picture. When you're done choosing options, click OK in the Style Builder.



Choices you make in the Style Builder aren't set in concrete. If you don't like a change you've made, you can undo it by pressing `Ctrl+Z`. Or, just reopen the style builder and change whatever options you want.

Adding Lines

If you want to add a horizontal rule to a page, expand the HTML category in the Toolbox. Then drag a horizontal rule onto the page, as illustrated in Figure 5-17. To style the line after placing it on the page, right-click the line and choose Style. Then, in the Style Builder, use Style and Color options under the Borders heading to choose the line's appearance and color.

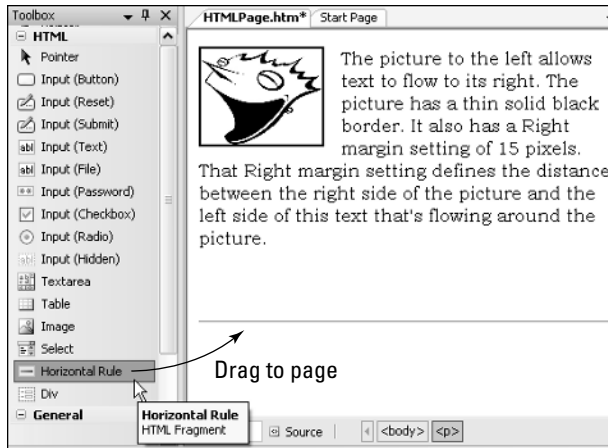


Figure 5-17:
Horizontal rule added to a page.

Editing in Source View

Everything you do in Design view gets translated to HTML (as well as CSS and ASP.NET) in your page. When you click the Source button at the bottom of the Design surface, you see all the tags. If you're familiar with (and like working in) HTML, you can add, change, and remove tags in Source view.

When you click one member of a pair of tags, like `<td>` or `</td>` for table cells, or `<p>` or `</p>` for a paragraph, both the opening and closing tags for the pair are boldfaced. That lets you see immediately what's contained within the paired tags.

When you're in Source view, you can use the + and - signs at the left side of the page to expand or collapse whole sections of the page. Doing so reduces clutter and allows you to focus on whatever portion of the page you're working on.

Selecting in Source view

You can select text and tags using all the standard techniques, such as dragging the mouse pointer through whatever you want to select. After you've selected something, you can move it by dragging it, or delete it by pressing the Delete (Del) key.

As you move the cursor about the page in Source view, tags that are near the cursor are represented by little buttons at the bottom of the Design surface. If you want to select a tag, its ending tag, and everything in between, just click the button. Or, right-click the button and choose Select Tag.

Alternatively, you can select everything between a pair of tags, and exclude the opening and closing tags. To do so, click the right side of the button that represents the tag, as at the bottom of Figure 5-18. Then choose Select Tag Content. (The Select Tag option has the same effect as just clicking the button — it includes the opening and closing tags in the selection.)

Figure 5-18:
Selection options on a button.



Typing tags and attributes

If you know HTML (and CSS) well enough, you can edit your page right in Source view. To add a new tag, just position the cursor to where you want to place the tag, and start typing the tag.

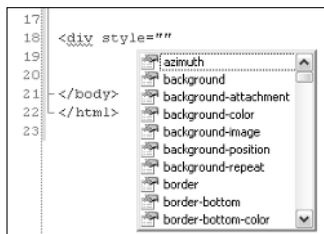
As soon as you start typing a tag, an IntelliSense menu appears, showing tags that match what you've typed so far. Rather than continue typing, you can double-click the tag name in the menu to finish what you've started typing.

To add attributes to a tag by typing, get the cursor just to the right of the tag text (and inside the `>` bracket if there is one) and press the spacebar to insert a blank space. The IntelliSense menu changes to show acceptable attributes for the tag you're typing. Again, you can continue typing, or double-click a name in the IntelliSense menu.

If you're validating your tags against a current XHTML spec, many attributes that you may be accustomed to may not show up in the IntelliSense menu. That's because XHTML has replaced many of the original attributes from HTML with CSS styles. When you use the Style Builder, you're actually adding CSS styles to your tags.

As an alternative to using the Style Builder, you can type CSS styling attributes right into the tag. Type **style=** (preceded by a blank space) into the tag. Then you can either just continue typing, or choose a CSS property from the IntelliSense menu that appears (Figure 5-19).

Figure 5-19:
IntelliSense
menu.



If you don't know a CSS from a cinderblock, don't worry about it at this point. Chapter 6 explains what CSS is about.

If you don't remember the exact CSS property:value pair you want to type, you can still use the Style Builder to design the element. Here's how:

- 1. First, make sure you click the tag you want to style, so the tag appears at the top of the Properties sheet.**
- 2. In the Properties sheet, click the `style` property, and click the Build button that appears.**
The Style Builder opens.
- 3. In the Style Builder, choose your formatting options then click OK.**

As soon as the Style Builder closes, the appropriate `style=` text is added to the tag. For example, Figure 5-20 shows how a `<div>` tag would look after choosing Silver as the background color in the Style Builder.

When you type the closing bracket of an opening paired tag, the closing tag for the pair is inserted automatically. For example, in Figure 5-20, the closing `</div>` tag was inserted automatically after I typed the closing angle bracket on the opening `<div>` tag.

Figure 5-20:

Paired

```
<div>...  
</div> <div style="background-color:Silver"></div>
```

tags with a
style.

To put text into the HTML element you just created, get the cursor right between the opening and closing tags. Then just start typing your text. Or, if you've already typed the text you want to put in the tag, just cut and paste it between the two tags.

Debugging HTML

All HTML and CSS that you type directly into Source view is validated against whatever version of HTML you've decided to use for your site. To review, or change, that selection, right-click any empty space in Source view and choose Formatting and Validation. In the Options dialog box that opens, click Validation under the Text Editor HTML heading. Then choose your HTML (or XHTML) preference from the Target drop-down list and click OK.

VWD won't be too fussy about rules as you type in Source view. But if you've created any problems while editing, you'll know about them as soon as you click the Design button to switch to Design view. If there are any problems, you'll see a message saying you can't go to Design view until you fix those errors. All you can do is click OK to stay in Source view and try to correct your errors.

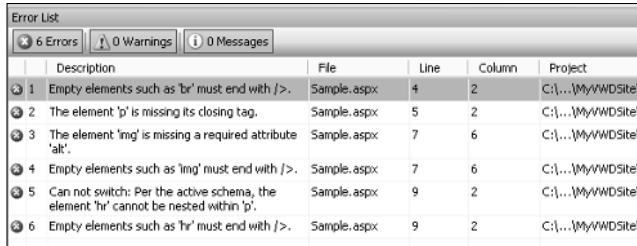


If you haven't made any significant changes to the page, you can just close the page and choose No when asked about saving your changes.

Any problems in a page that prevent you from switching from Source view to Design view are listed in the Error List pane below the Design surface, as in Figure 5-21. If that pane doesn't open, choose View ⇄ Error List from the menu bar to make it visible.

The severity of each error is marked by an icon. Errors that are marked with a red X have to be fixed before you can switch to Design view. Errors marked as *Warning* or *Message* are more like "suggestions" than actual errors.

Figure 5-21:
Sample
Error List
pane,
showing
errors.



The screenshot shows the 'Error List' pane with a summary bar at the top indicating 6 Errors, 0 Warnings, and 0 Messages. Below this is a table with columns for Description, File, Line, Column, and Project. The table contains six rows of error messages, each with a small icon in the first column.

	Description	File	Line	Column	Project
1	Empty elements such as 'br' must end with />.	Sample.aspx	4	2	C:\...\MyWebSite\
2	The element 'p' is missing its closing tag.	Sample.aspx	5	2	C:\...\MyWebSite\
3	The element 'img' is missing a required attribute 'alt'.	Sample.aspx	7	6	C:\...\MyWebSite\
4	Empty elements such as 'img' must end with />.	Sample.aspx	7	6	C:\...\MyWebSite\
5	Can not switch: Per the active schema, the element 'tr' cannot be nested within 'p'.	Sample.aspx	9	2	C:\...\MyWebSite\
6	Empty elements such as 'tr' must end with />.	Sample.aspx	9	2	C:\...\MyWebSite\

To quickly locate the source of an error, double-click its message in the Error list. The offending tag will be selected so you can make your change.

If you're not already an XHTML expert, you may want to stay out of Source view altogether, and stick to designing your page in Design view. That way, you won't make any errors that have to be corrected manually. However, if you're serious about being a Web developer, learning XHTML should be high on your to-do list. Things will be much easier after you understand what's going on with all those tags in Source view.

Chapter 6

Designing with Styles

In This Chapter

- ▶ Defining styles for your entire Web site
 - ▶ Creating CSS style sheets
 - ▶ Defining CSS style rules
 - ▶ Linking style sheets to your pages
 - ▶ Using CSS styles in your pages
-

If you want your Web site to have a unique character and be easy for visitors to get around in, it's important to maintain a consistent look and feel throughout all pages in your site. This involves thinking about what kinds of design *elements* you might use in your site. Examples of design elements include things like main headings, subheadings, body text, tables, picture borders, lines, and other items that might appear on pages throughout your site.

To maintain a consistent look and feel, it's best to predefine the exact appearance of all these items in *style sheets*. Doing so up front saves you a lot of time because you don't have to style every single heading, table, and picture as you add it to your page. Instead, you just format things normally and they automatically take on the appropriate appearance as you create them.

The real beauty of style sheets goes beyond consistency and ease of use to, well, preserving your sanity. If you ever decide to change the style of some element in your Web site, you don't have to go through every single page and make the change. You just change the style in the style sheet, and the new style is automatically displayed in every page. The technology you use to create style sheets goes by the name *Cascading Style Sheets*, or CSS for short.

Understanding CSS

CSS is a language that works in conjunction with HTML to define the exact appearance of any element in a Web page. You can use it in conjunction with the `style=` attribute in any HTML tag. When you choose options from the Style Builder introduced in earlier chapters, your selections get converted to a CSS `style=` attribute in the HTML tag.

If you want to give your site a consistent look and feel, you don't want to have to memorize and apply every style in every page. Instead, why not define the style once, in one place, and *apply* it everywhere? Now you're talking — and that's where Cascading Style Sheets come into play. They are the “one place” you define your styles for all your site's pages to share.

Before I talk about the sheets, let me talk about the “cascading.” The explanations in the official specs are daunting, to say the least. So let's take it from the top. Most HTML tags define elements in your Web page, things like the body of the page (between `<body> . . . </body>` tags), headings (`<h1> . . . </h1>` tags), paragraphs (`<p> . . . </p>` tags), and so forth. Some tags are contained within other pairs of tags. For example, the heading and paragraph (and the tags that define them) are between the `<body>` and `</body>` tags.

```
<body>
  <h1>I'm Heading</h1>
  <p>I am Paragraph. Heading and I are children of body.</p>
</body>
```

Here the heading and paragraph elements are both *children* of the body element, because they're contained within `<body>` and `</body>` tags. Like a family tree, the page body is the *parent* to the heading and paragraph elements, because both of those elements are contained within its opening and closing tags.

Now, suppose we use a CSS style to change the font of the body tag, as given here? What effect will that have? Behold:

```
<body style="font-family: 'Monotype Corsiva'">
  <h1>I'm Heading</h1>
  <p>I am Paragraph. Heading and I are children of body.</p>
</body>
```

The effect is that both the heading and the paragraph are shown in the Monotype Corsiva font in the Web browser. The reason is that both the `<h1>` and `<p>` tags inherit their default font from the parent element. Because the `<body>` element is the parent element to both the heading and paragraph, they both inherit its font style.

That brings up the main reason for “Cascading” Style Sheets: Certain stylistic elements *cascade* down through child elements. There's an “unless otherwise specified in the child tag” addendum to that rule. In other words, if a child tag has its own `style=` attribute, than that one child's element overrides the inheritance. Suppose (for example) we style the `<h1>` tag like this:

```
<body style="font-family: 'Monotype Corsiva'">
  <h1 style="font-family: 'Arial Black'">I'm Heading</h1>
  <p>I am Paragraph. Heading and I are children of body.</p>
</body>
```

When this code is viewed in a browser, the heading (only) is shown in the Arial Black font. The paragraph is still shown in the Monotype Corsiva font set in the `<body>` tag. That's because the new `style="font-family: 'Arial Black'"` style overrode the inheritance for that one element (the heading) only. The paragraph, which is its own separate element, has no `style=` attribute and hence still inherits the font specified in its parent element, the page body.

So, who wants to be bothered typing `style=` all over the place in HTML tags? Nobody. That's where style sheets come in. In the style sheet, you just define the rules of how each element will look, and all the elements take on the look you specified automatically.

The style sheet is the place where you define these rules. You don't use normal HTML tags and `style=` attributes in style sheets. Instead, you define CSS rules by using a different syntax: You simply type the name of the element you're designing, and then specify its style in a pair of curly braces. Here's an example of a simple CSS rule in a style sheet:

```
body {font-family: 'Bookman Old Style'}
```

Of course, there are lots of buzzwords that go with CSS. So let me first point out that every CSS rule (including the example above) consists of two major components:

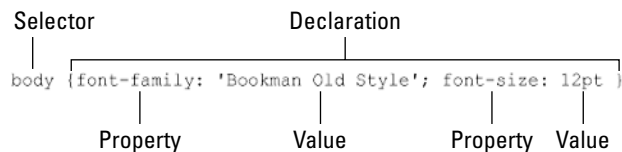
- ✓ **Selector:** The name that appears to the left of the first curly brace.
- ✓ **Declaration:** The text inside the curly braces.

In the example given here, the selector is the word `body` and the declaration is `font-family: 'Bookman Old Style'`. In English, the rule says *All pages in this Web site will use Bookman Old Style as the default font for body text (and everything that inherits the body tag's font)*. Brevity is the soul of CSS.

The declaration describes a style using one or more *property:value pairs* in which a colon (:) separates the property name from its value. In the example just given, `font-family` is the property and `'Bookman Old Style'` is the value of that property.

A rule can contain multiple *property:value* pairs, provided you separate them with semicolons (;). The rule shown in Figure 6-1, for example, has two *property:value* pairs in its declaration.

Figure 6-1:
A sample
CSS style
rule.



Rules don't have to be defined across a single line like the above example. In fact, VWD automatically arranges rules so each rule's property/value pair appears on a separate line. This makes it easier to see what's in the rule.

For example, here's yet another rule for the body style that defines the font type, font size, and background color for the page:

```
body
{
    font-size: 12pt;
    font-family: 'Bookman Old Style';
    color: navy
}
```

In the above example, the selector is on one line. The declaration is broken into five lines so you can clearly see the braces and the *property: value* pairs.

In VWD, you can create style sheets without knowing all the selectors, properties, and values, because you don't have to type anything. You can create your styles using the Style Builder instead. (Thank goodness.) First, you have to create a style sheet in which to put your CSS rules. The next section tells you how.

Creating a CSS Style Sheet

Creating a style sheet in VWD is fairly easy. If you want to keep your style sheets in a specific folder, you can start by right-clicking the project folder and the top of Solution Explorer and choosing New Folder. Name the folder whatever you like (I'll name mine, oh, what the heck, StyleSheets) and press Enter. Then, to create a new style sheet, follow these steps:

- 1. In Solution Explorer, right-click the folder in which you want to place the style sheet and choose Add New Item.**
- 2. In the Add New Item dialog box, click Style Sheet.**
- 3. In the Name box, type a name for your style sheet.**

In Figure 6-2 I've named mine `MyStyles.css`.

- 4. Click Add.**

A new style sheet opens in the CSS Editor, at first just showing an empty rule for defining the style of the HTML body tag. The Style Sheet toolbar also opens, as shown at the top of Figure 6-3.



If your CSS toolbar doesn't open somewhere, choose `View ⇄ Toolbars ⇄ Style Sheet` from the menu bar.

Figure 6-2:
Creating a
new CSS
style sheet
named
MyStyles
.css.

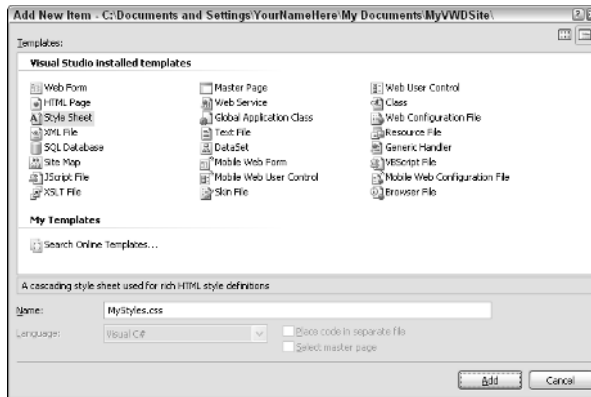
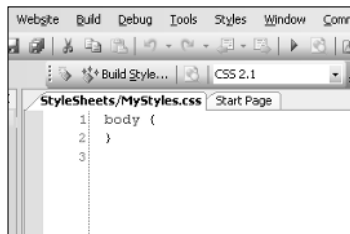


Figure 6-3:
The Style
Sheet
toolbar and
part of
MyStyles
.css.



If you're already into CSS and have a preference, you can use the drop-down list on the Style Sheet toolbar to choose with which version of CSS you want your sheet to comply. If you don't have a preference and just want to go with the latest version, choose CSS 2.1 (as in Figure 6-3).

Creating Style Rules

A style sheet can contain any number of rules — of several different types. The sections that follow look at the different types of rules and how to create them.

Creating CSS element styles

A CSS element style is a rule that defines the style of a built-in HTML tag like `<p>` or `<body>` or `<h1>`. These rules are the easiest to create and the easiest to use (in most cases). The predefined elements cover most of the design elements needed for a Web page. So rather than reinvent the wheel, you can set a style for those elements and just use the elements as you would normally.

In a CSS style sheet, you follow these steps to create an element-rule selector:

1. **Click the Add Style Rule button in the CSS toolbar, or right-click in the CSS style sheet and choose Add Style Rule.**

The Add Style Rule dialog box opens.

2. **Choose Element, and then choose the HTML element for which you want to design a style.**

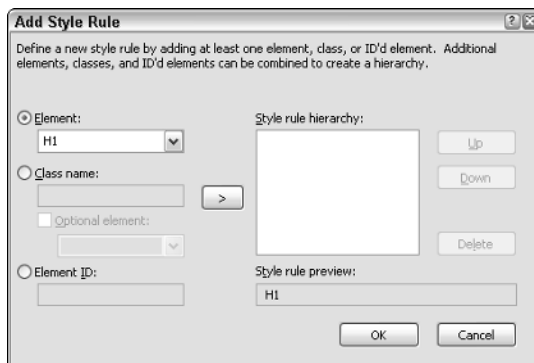
In Figure 6-4, I chose the H1 element, allowing me to define the style of all text marked with `<h1> . . . </h1>` tags in my pages.

3. **Click OK.**

A rule selector appears with empty curly braces, as in the example given here:

```
H1
{
}
```

Figure 6-4:
Defining
a style rule
for the H1
element.



With the selector and curly braces in place, you can use the Style Builder described later in this chapter to define the style of the element.



If you already have a style sheet from a previous Web site, you can just copy and paste that sheet's contents into the new sheet you've created in VWD.

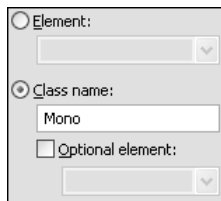
Creating CSS class selectors

You don't have to apply a style to all instances of an HTML element. You can create "special circumstance" rules that are applied only as needed. For example, you might have to create two types of tags for table cells: one for regular table cells, and one for the column headings across the top row of the table.

To create “special circumstance” rules, you define CSS *class selectors*. Unlike an element selector, a class selector doesn’t have to match an HTML tag. You can give the class selector any name you want — just avoid using spaces and punctuation in the name. Keep it short and simple.

You create a class selector as you would an element selector. Click the Add Style Rule button or right-click the page and choose Add Style Rule. In the Add Style Rule dialog box, choose Class Name. Then type the name of the class in the text box, as in the example shown in Figure 6-5.

Figure 6-5:
Creating a
CSS class
selector.



Click OK, and the class appears in the style sheet looking like this:

```
.Mono  
{  
}
```

The leading dot that VWD adds to your name isn’t actually part of the name. It’s just a signal to the Web browser, indicating that the name that follows the dot is a class name rather than an HTML element.

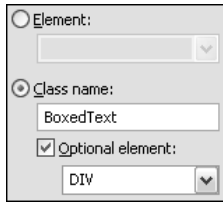
You can also assign a style class to a particular HTML element, and use that style class with only certain instances of the element. For example, you might create a DIV (division) class that creates sidebar text, similar to the sidebars you see in this book. In a style sheet, you create such a selector by joining the HTML element name (the same name you use inside an HTML tag) followed by a dot and whatever name you decide to give your class.

Again, you use the Add Style Rule dialog box to create the rule. Choose Class Name and type in a name of your own choosing for the class. Then select the Optional Element check box, and choose an HTML element from the drop-down list. For example, in Figure 6-6, I’m about to create a new CSS class named `DIV.BoxedText`.

Click OK, and the result is as follows:

```
DIV.BoxedText  
{  
}
```

Figure 6-6:
Defining
a CSS
element
class
selector.



Now all that's left to do is make sure the cursor is between the curly braces, and use the Style Builder to define the style of the rule. The next section tells you how.

Defining Rules with Style Builder

Manually typing the declaration for a rule isn't easy. The syntax rules are strict, and there are a lot of property names to remember. The Style Builder offers a much better alternative. The first step, or course, is to create the selector (you know — the name and curly braces). The next step is actually two:

1. **Position the cursor just to the right of the opening curly brace of the selector you want to style.**
2. **Click the Build Style button in the Style Sheet toolbar, or right-click near the cursor and choose Build Style.**

In Figure 6-7, for example, I'm about to build a style for the `body{}` selector in `MyStyles.css`.

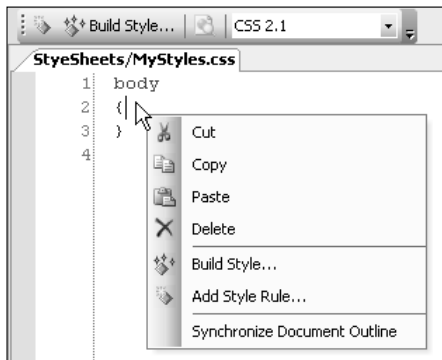


Figure 6-7:
Preparing to
build a style
for the
`body{}`
selector.

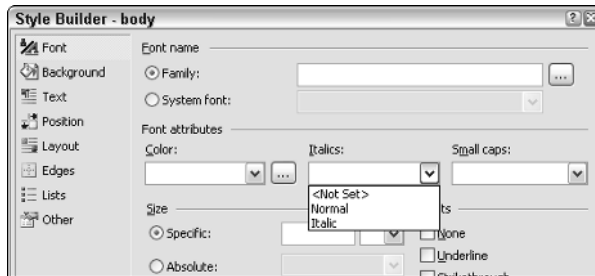
After you've chosen Build Style, the Style Builder dialog box opens. Figure 6-8 shows the top half of the Style Builder. Here you can add all the styles you like.

Setting to <not set>

Many of the drop-down lists you encounter in the Style Builder will have a <not set> option, like the Italics drop-down list in Figure 6-8. It may seem odd that you have the choice to set an option to “not set,” but there is a reason for it: If you previously set the option to a specific setting, choosing <not set> ensures that the old setting is removed.

In the rule that the Style Builder creates, you won't see <not set> anywhere because choosing <not set> removes the *property: value* pair from the rule. That means the element you're styling will inherit the style from its parent element.

Figure 6-8:
Here's what
part of the
Style
Builder
looks like.



The sections that follow discuss the most commonly used options in the Style Builder. As with HTML, CSS is a book-length topic in its own; an exhaustive discussion of styles would surpass my page count (and probably your patience). But a quick tour of the most common style options can give you a sense of how you might use them in your own Web sites. Stay tuned.



For more information about the options in the Style Builder category you're viewing, press Help (F1).

Styling fonts

The Font category of the Style Builder lets you define the font of the selector (the item you're styling). It's important to know that fonts come from the client computer, not from your computer. In the person doesn't have the font specified in your CSS rule, their Web browser will automatically substitute another font.

To gain some control over how the client PC chooses substitute fonts, specify multiple fonts in order of preference. Generic fonts are good for this purpose, because they allow the client PC to choose a font that at least resembles your preferred font. To specify fonts in the Style Builder, choose Family under Font Name in the Style Builder. Then click the Build button.



A generic font is just a general font style without a specific typeface. For example, Arial and Helvetica are both sans-serif fonts used for headlines. If you specify Arial as the preferred font, and sans-serif as the Generic font, Arial will be used wherever possible. Otherwise, Helvetica (or some other sans-serif font) will be used instead.

In the Font Picker, first choose the font you most prefer, and then click the >> button to move that font name to the Selected Fonts list. Any additional fonts you choose after the first will be used only if your preferred font isn't available on the client PC. Figure 6-9 shows an example in which I've chosen Bookman Old Style as the preferred font, and a generic Serif font as the alternate. Click OK after making your selections.

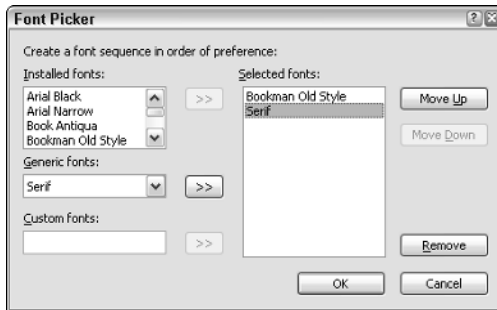


Figure 6-9:
The Font
Picker
dialog box.

To choose a color for your font, pick a color name from the Color drop-down list. Or click the Build button to the right of the color option. In the Color Picker dialog box that opens, click any tab, and then choose your color and click OK.

There are several ways to size a font:

- ✔ **Set a specific size, in points (or some other unit of measure):** You might want to do this when defining the default font for body text (in the `body { }` selector in the style sheet). To set a specific size, choose Specific. Then type in a size and choose your unit of measure.
- ✔ **Set an absolute size:** This option defines the size as a value, ranging from XX-small to XX-large. The exact size of the text, in points, is defined by settings in the visitor's Web browser.

Absolute sizes are defined by the visitor's Web browser, with Medium being equal to body text size. The exact size varies. For example, in Microsoft Internet Explorer you can choose View↔Text Size to set a general size for all text on the page. Text you define using an Absolute size will be sized according to which Text Size a visitor chooses from that menu.

To get a sense of how the absolute sizes vary relative to one another, choose Choose a Size from the Absolute drop-down list. The sample text at the bottom of Style Builder shows you an example of your choice.

- ✓ **Set a relative size:** This defines the size of the text relative to neighboring inline text, either smaller or larger.

Relative size refers to the size of the parent element's size. The parent element is usually text in the same sentence of a paragraph. This setting makes the text a little smaller, or a little larger, than whatever size that neighboring text happens to be.

The option for boldface reflects XHTML's preference for using the CSS `font-weight` property rather than `...` tags for boldface. You can choose how bold you want boldface to be using that property. If you want the standard boldface, choose Bold from the Absolute drop-down list. Otherwise choose Relative and then choose either Lighter or Bolder.

By and large, the other items in the Fonts category are self-explanatory. If they still seem a little obscure, just choose one and have a look at its preview sample at the bottom of the Style Builder.

Styling the background

You can choose a background color or picture for the item you're designing. Optionally, you can choose a picture to use as a watermark for the background. Click Background in the Style Builder to get to the Background options.



A *watermark* is a pale image that appears to be imprinted on the page, behind text and other items on the page.

To specify a background color, choose a color name from the Background Color drop-down list. Or click the Build button next to the Color option and choose a color from the Color Picker.

If you have a picture you'd like to use as a watermark, click the Build button next to the Image option and choose the picture you want to use. If it's a small picture like a logo, you can choose a Tiling option to have the image repeated down, across, or all over the page. The scrolling options let you choose whether the background image scrolls with text, or remains fixed as the text scrolls over the image.

The Position options determine the starting position of the background image. The effect of the Position option depends on how the Tiling option is set. The preview at the bottom of the Style Builder shows how your current selections will look on a page. Figure 6-10 shows an example with a background color, image, and some image options selected. The preview shows how those selections look.

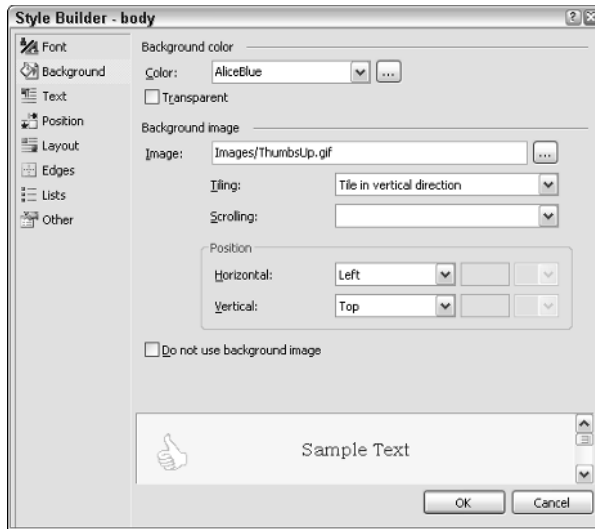


Figure 6-10:
Background
options in
Style
Builder.



In Figure 6-10, the background color shows through the background image because that image file has a transparent background. If you use a picture with an opaque background and tile that image in all directions, its background will hide whatever color you choose for the page background.

The options in the Background category of the Style Builder relate directly to the CSS properties that control background — `color`, `background-image`, `background-repeat`, `background-position`, and `background-attachment`.

Styling text alignment and spacing

Clicking the Text button in the Style Builder takes you to styling options for text (Figure 6-11). Under the Alignment heading, the Horizontal option lets you choose how you want the text, or item, aligned between the margins of the parent object. For example, if you're styling a table cell (`<td>` element), and set the horizontal alignment to Center, any text or picture you place in the cell is centered within the cell.

The Vertical alignment option offers several settings, not all of which apply to all types of elements. The `sub` and `superscript` elements, for example, apply to inline text only. Thus, they only apply to *inline elements* — those that don't cause text, a picture, or a table to start on a new line. Examples of inline elements that don't cause line breaks include ` . . . ` for boldface, as well as ` . . . ` and ` . . . `.

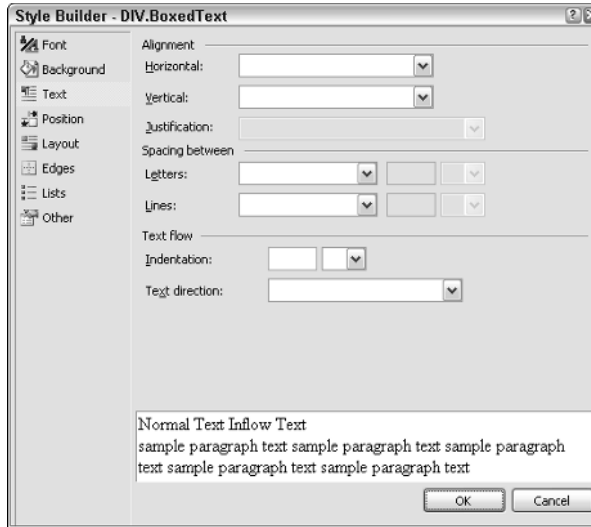


Figure 6-11:
Text
category of
options in
Style
Builder.

In Figure 6-12, the top two lines show examples of using `superscript` and `sub` vertical alignments. The `superscript` alignment raises the text from the normal flow. The `sub` alignment lowers it, as in a subscript.

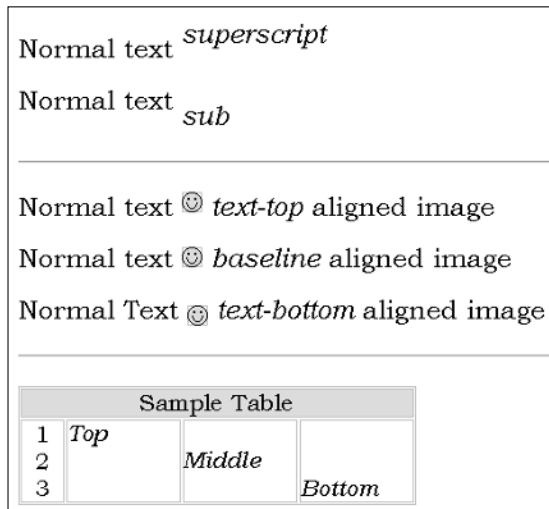


Figure 6-12:
Examples of
vertical
alignment
options.



Unlike the HTML `<sup>` and `<sub>` tags, the vertical alignment options don't change the size of text. They change only its position relative to other text in the same line.

If you're styling an `` tag used to show a very tiny picture, use the *text-top*, *baseline*, and *text-bottom* vertical alignment options; these give the picture a precise vertical position relative to inline text. The middle example in Figure 6-12 shows examples of vertically positioning a tiny happy-face image.

If you're styling a block element that contains text and has some height to it, use the Top, Middle, and Bottom vertical alignment options to align text within the box. In Figure 6-12, the Top, Middle, and Bottom vertical alignment options are applied to cells in a table where the row is three lines tall.



When you're styling a table, keep *inheritance* — the relationship between parent and child tables — in mind. Any alignment options that you want to apply to the table as a whole should be defined in the table element (the `<table>` tag in HTML), because each cell in the table is a child element, defined by `<td>` tags, within the table. Therefore, any style you define for the table will be inherited by all cells in the table.

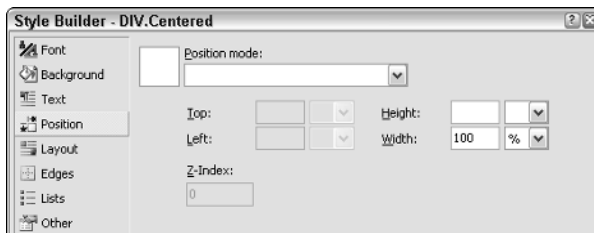
Of course, you can always override the style in any cell or group of cells. When you change the setting of a single cell, you do so only for that one cell, that one `<td>` tag. Other cells still inherit the characteristics of the parent `<table>` tag.

The spacing and flow options in the Text category of the Style Builder are largely self-explanatory. As soon as you choose an option, the preview sample given here shows you the effects of your choice. The words “Normal Text” in the preview never change, so you can compare the results of any change you make to that text. If you don't like a change you've made, choose `<not set>` to cancel your selection, or delete whatever you typed into an option.

Styling position

The Position category of Style Builder, shown in Figure 6-13, offers options for sizing and positioning pictures (the `` tag), tables (the `<table>` tag), and block elements that contain text like the `<div>...</div>` (division) tags.

Figure 6-13:
The Position
category in
the Style
Builder.



From the Position Mode drop-down list, you can choose from the following options:



✓ **Position in normal flow:** The element is “glued” to its neighboring text (usually a paragraph). When the paragraph moves, the text moves with it. This is the default for most blocks; they end up in normal flow if you don’t specify otherwise in a style.

The “palm tree” and “laughing kid” pictures in Figures 5-14 and 5-16 of Chapter 5 are in `` tags that are positioned in normal flow.

✓ **Offset from normal flow:** Same as above, but the item can be offset from its default position. Choosing this option enables the Top and Left options; use those to specify the offset. For example, entering 10px for Top and 5px for Left would move the element down 10 pixels and 5 pixels to the right, where it would have been positioned by default.

✓ **Absolutely position:** Places the block at a specific position on the page. The block is not “glued” to any neighboring text. When you insert or delete text above the block, text below that point moves around the object rather than dragging the block with it.

Choosing the Absolute position options enables the Top and Left options. Use them to define where you want to place the upper-left corner on the page. You can express the measurement in pixels, percent, points, or whatever is most convenient for you.

For example, if you absolutely position the top of a block element to 40%, and the left to 10px, the top-left corner of the block will be 40% of the distance to the bottom of the page, and 10 pixels away from the left margin. The block won’t change position as you add or delete text above it.

If you choose Absolute position, you can also set a *Z-Index* for the block. A *Z-Index* is a number that defines the *layer* on which a block element is placed. Think of a stack of paper, with each sheet of paper as a layer. The sheet at the bottom of the stack is at layer 0 (or *Z-Index* 0). The next sheet up is at layer 1 (*Z-Index* 1), and covers the sheet below it. The next sheet up is at layer 3 (*Z-Index* 2), and it covers layers below it.

Using *Z-Indexes* is tricky, because items with *Z-Indexes* greater than zero can cover content on your page. That sort of thing can really bother visitors who are trying to read your page. So you want to stay away from setting any *Z-Index* unless you have a specialized application for it in your site.

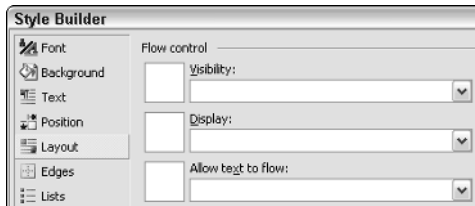
The Height and Width options in the Style Builder Position options let you define the size of the image, table, or other block element you’re styling. You can define the size in pixels, a percent of the page width (or height), or any of several other units of measure.

Styling layout

In the Layout category of the Style Builder, you find options that control how elements appear on your page in relation to one another. Some are beyond the scope of this book. But the first three Flow Control options (shown in Figure 6-14) are right up our alley.

Figure 6-14:

Some options in the Layout category of Style Builder.



- ✔ **Visibility:** This option controls whether or not the block is visible on the page. For instance, you might want to make an element initially invisible, and then have it appear when the visitor clicks a link or button. The Visibility options are Hidden (the element isn't visible on the page) and Visible (the element is visible). The `<not set>` option is the same as Visible.
- ✔ **Display:** This option determines whether the item is displayed as a block element or inflow element. If treated as a block element, the item is placed in its own box on the page, like a table or image. If treated as an inflow element, the style is applied inline. Boldface and italics are good examples of inline elements. Both apply their styles to text without disrupting the flow of text.
- ✔ **Do Not Display:** This option omits the element from the page altogether — but it's not the same as the Invisible option (which sends the element to the client PC but hides it from view, perhaps temporarily). The Do Not Display option prevents the element from being sent to the browser. That reduces the download time. A good example would be sending content to visitors selectively, based on the capabilities of the visitor's Web browser.
- ✔ **Allow Text To Flow:** These options determine whether (and where) text flows around the block element you're styling. Figure 5-14 in Chapter 5 showed an example using images. You have three basic choices:
 - Choosing Do Not Allow Text On Sides prevents any text from wrapping around the element. The element appears on its own line.
 - Choosing To The Right causes the element to align to the left margin, so text can flow down the right side of the element (as with the laughing boy in Figure 5-16 in Chapter 5).



- Choosing To The Left causes the item to align to the right margin, so text can flow down its left border.

If you want a *really* thorough description of CSS layout properties, see Chapters 8 through 10 in the CSS 2.1 spec at www.w3.org/TR/CSS21.

Styling boxes and borders

Many design elements, like pictures, tables, and chunks of text enclosed in `<div>...</div>` tags, form boxes on your page. The Edges category in Style Builder allows you to define the spacing outside the box, inside the box, and the appearance of the borders around the box. If you're defining the style of a box that contains text or an image, the settings apply as follows:

- ✓ **Margins:** Defines the margins outside of the box (how near neighboring text can get to the box).
- ✓ **Padding:** Defines margins inside the box (how near text inside the box can get to the box border).
- ✓ **Borders:** Defines the style, width, and color of the border surrounding the box.

Figure 6-15 illustrates where the Margins and Padding settings apply to a box that contains text.

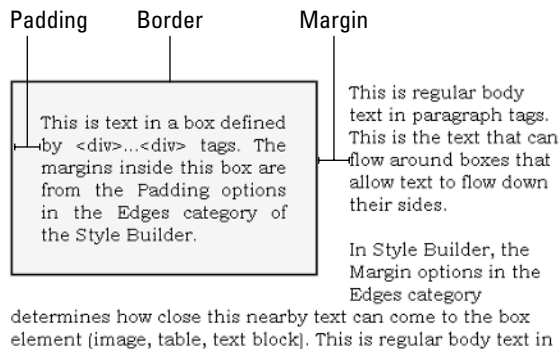


Figure 6-15:
Margins,
padding,
and borders
for a box.

A table's margins are, in effect, little gaps between cells, as shown in Figure 6-16. The padding puts space between a cell's inner border and the contents of the cell. The Borders options control what kind of lines go around the table and between the cells.

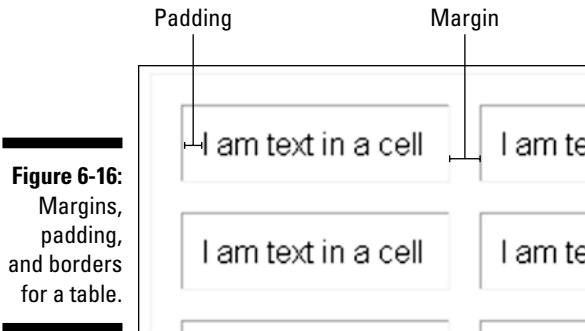


Figure 6-16:
Margins,
padding,
and borders
for a table.

When designing borders, you can choose to define all the borders at once, or just the left, right, top, or bottom margin.

You can also collapse the vertical borders in a table so there's only one line and no gap between cells vertically. This only works if you're defining the style characteristics of a table element (that is, using a `<table>` tag). To collapse vertical borders in the table, click Other in left column of the Style Builder. Then, under the Tables heading, choose Collapse Cell Borders from the Tables drop-down list.

Saving Style Builder choices

When you've finished choosing options from the Style Builder, just click OK. Your selections are translated into appropriate CSS *property: value* pairs and inserted between the curly braces for the rule. Here's a general example:

```
DIV.BoxedText
{
  border-right: navy thin solid;
  padding-right: 25px;
  border-top: navy thin solid;
  padding-left: 25px;
  float: left;
  padding-bottom: 25px;
  margin: 0px 25px 15px 0px;
  border-left: navy thin solid;
  padding-top: 25px;
  border-bottom: navy thin solid;
  width: 30%;
  background-color: #ffffcc;
}
```

To change your selections, right-click anywhere between the curly braces and choose Style Builder again. Make your changes and click OK. Remember, your style sheet can contain as many, or as few, style rules as you like.



All those *property: value* pairs that Style Builder creates are straight from the online CSS specification, available at www.w3.org/TR/CSS21.

Saving a CSS style sheet

When you've finished working with a style sheet, just close and save it as you would any other document. The styles you defined won't be applied to anything until you link the style sheet to a page (or Master Page), as discussed next.

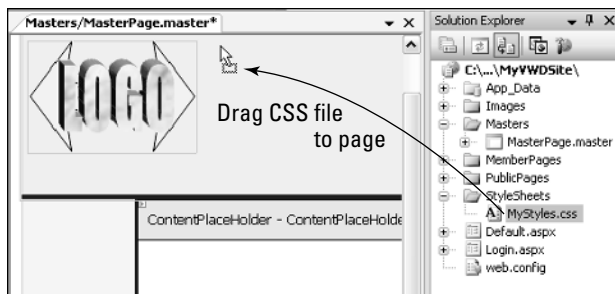
Linking to a Style Sheet

The rules you create in a CSS style sheet are applied only to the pages to which you link the sheet. If you've created a Master Page, you can link the style sheet to the Master Page. The styles automatically carry over to all content pages that use the Master Page.

The easy way to link to a style sheet is just to drag its icon from Solution Explorer into the page you're editing. For example, to add a CSS link to a Master Page, you simply open the Master Page (by double-clicking its icon in Solution Explorer), drag the CSS file's icon from Solution Explorer to the upper-left corner of the page you're editing, and drop it there.

Figure 6-17 shows an example: I'm dragging a style sheet named `MyStyles.css` into an open Master Page named `MasterPage.master`.

Figure 6-17:
Link a style sheet to a Master Page.



To use your style sheet in an HTML page (or an `.aspx` page that doesn't use the master), just open that page in Design view and drag the style sheet's icon from Solution Explorer onto the page.

Unless there are styles in the style sheet that apply to tags already in the Master Page, you won't notice any difference. But if you switch to Source view, you'll see that VWD has inserted a link to the style sheet between the `<head>` and `</head>` tags near the top of the page. That link will look something like this example:

```
<link href="../../StyleSheets/MyStyles.css" rel="stylesheet" type="text/css" />
```

Keep in mind that only pages that have that tag near the top use the styles in the style sheet. If you create a new page from scratch, and don't specify a Master, drag the CSS file over to the page to create the link.

Using Styles in a Page

After you've linked a style sheet to a page, HTML element styles will be formatted automatically. For example, if you styled the `body{ }` element, then body text in the page takes on the style you defined immediately.

As you format text in the page, any new styles will come into play as you use the element. For example, let's say you created a style for Heading 1 elements in your CSS style sheet, like this:

```
H1
{
    font-size: 16pt;
    font-family: 'Arial Black';
    font-style: italic;
}
```

Any text in the document that is already formatted as H1 will automatically be displayed in the new style. That is, any text between the `<h1>` and `</h1>` tags in the Web page will be displayed in 16 point Arial Black italic font.

Exactly how you use a style rule to format new or existing content on the page depends on which type of selector defines the rule. I'll start with the easiest one, in which the selector's name matches an HTML tag (for example, as H1 or BODY).

Applying CSS element selectors

A CSS *element selector* is any CSS rule whose name does not contain a dot. The name of the selector matches the name inside the HTML tags of the element. Applying such a selector takes no real effort at all. You just apply the element as you normally would.

For example, suppose the currently linked CSS style sheet contains the H1 style shown earlier. In the current document, you select a chunk of text, and then choose the Heading 1 style from the Block Format drop-down list on the Formatting toolbar, as in Figure 6-18. The style is applied to the selected text.

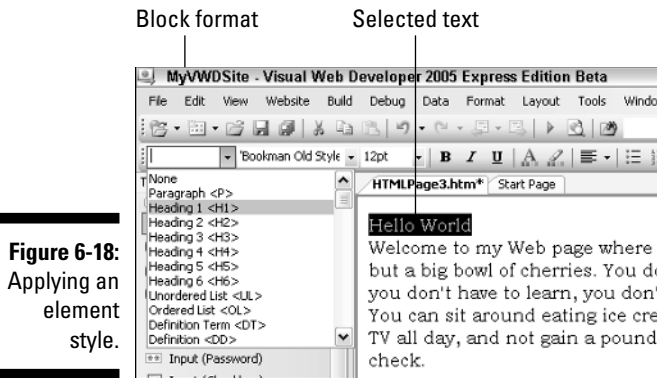


Figure 6-18:
Applying an
element
style.

In Source view, the formatted text is contained in the usual pair of HTML tags, like this:

```
<h1>Hello World</h1>
```

The words `Hello World` take on the formatting defined by the H1 style rule in the style sheet. You don't have to do anything special to make that happen. The connection between `<h1>` in the page and the `H1 { . . . }` style rule in the style sheet is built-in and automatic. If you change the `H1 { }` style rule in the style sheet, the change shows up in all pages automatically.

Applying CSS class selectors

CSS class selectors aren't associated with any particular HTML element. In a style sheet, their names are preceded by a dot, as in this `.Mono` class selector:

```
.Mono  
{  
    font-family: 'Courier New' , Monospace;  
    font-style: normal;  
    color: navy;  
}
```

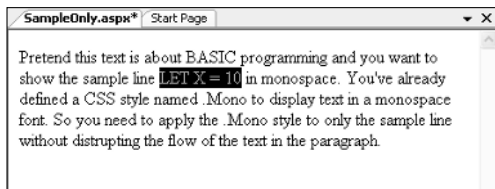
Using CSS class selectors in VWD is a bit of a pain, because their names don't show up in IntelliSense menus or any drop-downs anywhere. You have to remember the names and type them yourself. For an inline style like the `.Mono` example, your best bet would be to enclose the text to be formatted in ` . . . ` tags in Source view and specify the class name in the `` tag, as follows:

```
<span class="Mono">text to format goes here</span>
```

Note that within the `` tag you don't include the leading dot, just the name. The class name must be enclosed in single or double quotation marks.

You'll need to work directly in Source view to type the tags. But it will be easiest to find the text if you first select that text in Design view. For example, let's say in Figure 6-19 you want to display the sample code `LET X = 10` in monospace. First, select that text as shown in Figure 6-19 by dragging the mouse pointer through only that text.

Figure 6-19:
Selecting
text in
Design
view.



To switch to Source view, click the Source button at the bottom of the Design surface. The text you selected in Design view is still selected in Source view, making it a little easier to find among all the other stuff that appears in Source view.

Next, click a space just to the left of the first character you want to format, then manually type in the `` tag, as follows:

```
<span class = "Mono">
```

VWD automatically adds the closing `` tag right after the opening tag. You need to move that closing tag so that it's after the text to which you want to apply the style. A simple way to do that would be to cut and paste the text so that it's between the two tags. For example, in Figure 6-20 I've moved the sample code `LET X = 10` so it's between the ` . . . ` tags.

Figure 6-20:
Applying a
CSS class
selector to
an ``
tag.

```
11 | <div>
12 |   Pretend this text is about BASIC programming
13 |   <span class="Mono">LET X = 10</span> in monos
14 |   text in a monospace font. So you need to appl
15 |   line without disrupting the flow of the text
```

When you switch back to Design view, only the text within the tags will be formatted with the Mono style.



If you switch to Design view and the text isn't formatted as expected, either you forgot to link the style sheet to the current page, or you misspelled the class name in the Style tag.

Applying element class selectors

Element class selectors have a dot embedded in the name. The first part of the name defines which tag you use to apply the style. For example, here is an element-class selector named `ColHead` that can only be applied to table cells (`<TD>` tags).

```
TD.ColHead
{
    font-weight: bold;
    font-family: Arial;
    background-color: #ffffcc;
    border-bottom: navy thin solid;
}
```

To use an element class like `TD.Colhead`, you have to apply the class to text that's already formatted by the HTML element, `TD` in this example. Because `<TD> . . . </TD>` tags define table cells, you have to start, in Design view, by clicking on text that's already in a table cell.

You can take a shortcut and select the items you want to format. For example, to apply the `TD.ColHead` class selector to all the cells in a row, select the row as in the example at the top of Figure 6-21. The Properties sheet won't show a tag name when you have multiple elements selected. But you can still type the class name, `ColHead`, in the `Class` property for the selection, as in the center of that same figure. Press the `Tab` key to complete the entry, and the selected cells will take on the style, as in the bottom of that same figure.



Figure 6-21:
Applying
TD.ColHead
to multiple
cells.

Using DIV styles

It's worth mentioning the HTML `<div>...</div>` tags at this point, because they're relevant to CSS styles. A *div* is a box or “division” of text, set off from the normal flow of body text.

The `<div>` tag is to block styles what `` is to inline styles. An empty pair of `<div>...</div>` tags creates a new box in which you can place text, tables, pictures, or whatever. However, text inside the box inherits its font and virtually everything else from its parent element, most likely the `<body>` tag for the page.

The main reasons I mention the `<div>` element is because many of the old ways of aligning things — for instance, the `align=` attribute — are being phased out in favor of using `<DIV>` tags.

As an alternative to manually creating and formatting a pair of `<div>...</div>` tags each time you put a centered element on a page, you could create a CSS style that already defines how you want centered elements to appear on the page. For example, the CSS style below, named `DIV.Centered`, ensures that any element between its tags is centered on the page, and that no text flows around the centered element.

```
DIV.Centered
{
  background-color: transparent;
  text-align: center;
  width: 100%;
  float: none;
  clear: both;
}
```

In the Style Builder you'd choose the following categories and options:

- ✓ **Background:** Choose Transparent.
- ✓ **Text:** Choose Horizontal, Centered.
- ✓ **Position:** Choose Width property and set it to 100%.
- ✓ **Layout:** Set Allow Text To Flow to Don't Allow Text On Sides. Set Allow Floating Objects to Do Not Allow.

To use the style in a linked page, first make sure the page you're editing is linked to the style sheet. Then drag the HTML Div control from the toolbox onto the page. An empty box appears and `<DIV>` appears at the top of the Properties sheet. Type the class name into the `Class` property, and press Tab. The box won't change immediately. Figure 6-22 illustrates the procedure.

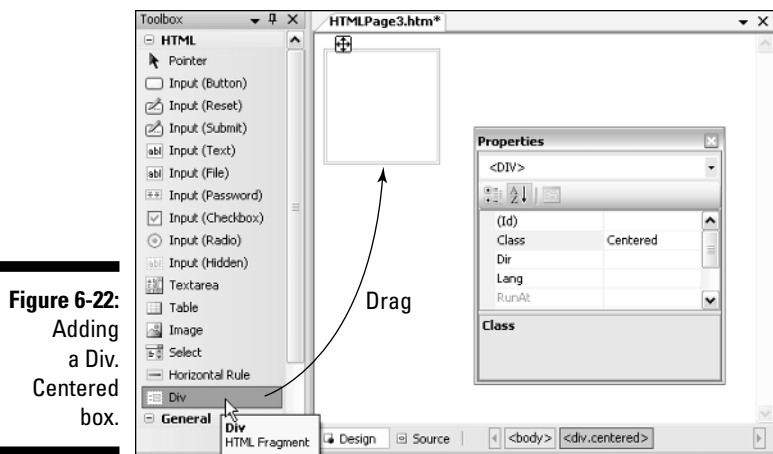


Figure 6-22:
Adding
a Div.
Centered
box.

To center an image on the page, drag its icon from Solution Explorer into the box. To center a table on the page, click inside the box before you choose `Layout>Insert Table` from the menu bar. Then create the table normally.

In the VWD Design view, the `<DIV>` box will be visible as gray lines around the object inside. But in the Web browser, no such lines are visible and the item is just centered on the page.

In the HTML for the page, a picture appears as an `` tag within `<div>` tags as in the following example:

```
<div class="Centered">  
      
</div>
```

The CSS 2.1 Specification

This chapter has been about using CSS style sheets in VWD, presuming you're at least vaguely familiar with CSS already. There's not enough room in this book to cover the entire CSS specification. That's a topic in itself you could learn in a class, from a book, or from the online specification and tutorials.

Like the HTML specs, the CSS specs are available at the World Wide Web Consortium Web site at www.w3c.org/. The following short list shows pages specifically relevant to CSS 2.1. Check them out. Any time you invest in learning CSS is time well spent.

- ✔ **Starting with HTML + CSS:** www.w3.org/Style/Examples/011/firstcss
- ✔ **Introduction to CSS 2.1:** www.w3.org/TR/CSS21/intro.html
- ✔ **CSS 2.1 Specification:** www.w3.org/TR/CSS21/

Chapter 7

Working with ASP.NET Controls

In This Chapter

- ▶ What is ASP.NET, anyway?
 - ▶ Using server controls in your pages
 - ▶ Allowing users to create their own accounts
 - ▶ Creating a login page
 - ▶ Letting users change their passwords
-

The big difference between creating a regular Web site with HTML and a dynamic Web site with Visual Web Developer mostly boils down to ASP.NET controls. ASP stands for Active Server Pages. The .NET refers to the Microsoft .NET Framework, the home of thousands of controls for all types of programmers, not just Web developers.

Some readers may already know what all that means and what it's all about; some may not. For those who don't, we start this chapter with a quick overview of ASP.NET and why it's important for Web developers.

What Is ASP.NET?

Glad you asked. ASP.NET is essentially a set of controls for a building dynamic, data-driven Web site. To understand what that means, let's start with a (hopefully familiar) example of searching the Web using a search engine like Google.

Suppose you go to `www.google.com`, type in something like *1966 Ford Mustang convertible parts*, and press Enter. A moment later, Google sends back a page with links to a bunch of Web sites that contain those words (not to mention quite a few relevant ads). Where did that page come from? How does that work?

Certainly Google didn't have a Web page already made up and waiting, just in case someone happened to search for "1966 Ford Mustang convertible parts". There are a near infinite number of word combinations that people might search for. And there's no way to create a near infinite number of Web pages to cover every conceivable combination of words people might search for.

Instead, Google's Web server has to take whatever words a user submits, search its database of 8 billion or so links for pages that contain those words, then *create* a page that contains appropriate links to send to the user's Web browser.

In other words, Google's Web server had to take an *active* role in creating the page it sent to your Web browser.

That is the very essence of a dynamic, data-driven Web site. The pages that Google sends out from its Web server are *dynamic* in that each page contains only links that match the words the user searched for. Those results are *data-driven* in that all those links are stored in a database at Google's site.



I'm not saying the Google was created with Visual Web Developer. Google was created long before Visual Web Developer existed. But the concept of having the Web server take an active role in creating a Web page is all that matters for this analogy. Likewise, we won't be building a search engine in this book. The only analogy that matters is the fact that Google's Web server plays an active role in creating the Web pages that get sent to people who conduct searches.

Making your Web server take an active role in creating the pages it sends is what ASP.NET is all about. Active Server Pages (Web pages with an `.aspx` extension) are basically Web pages that contain Active Server Controls. But you can't send pages that contain server controls directly to clients; clients can't execute server controls. The server uses data from the database and information in the `.aspx` page to determine exactly what HTML is needed on the client PC. Then the server creates the appropriate page and sends it to the client. Figure 7-1 illustrates the basic idea.

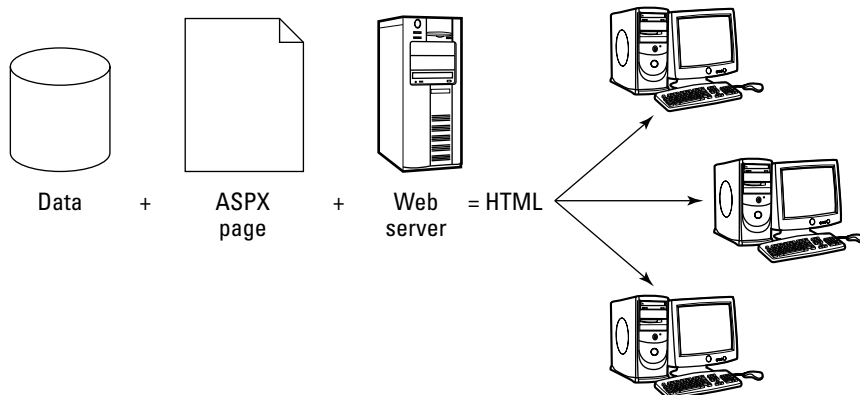


Figure 7-1:
Using data
and `.aspx`
pages to
create
HTML.

The .NET Framework is an enormous collection of tools for programmers (and, like “the world and all it contains,” not a subject we have to get into right now). Suffice it to say that all ASP.NET Web server controls you find in VWD are members of the .NET Framework. And as such, complete documentation about any server control is available from the .NET Framework online documentation.

In Visual Web Developer, the Toolbox shows the names of commonly used .NET Server controls, categorized into groups like Data, Validation, Navigation, Login, and so forth. These tools are visible only when you’re editing a Web form (.aspx page). The server controls are hidden when you’re editing an .HTML page.

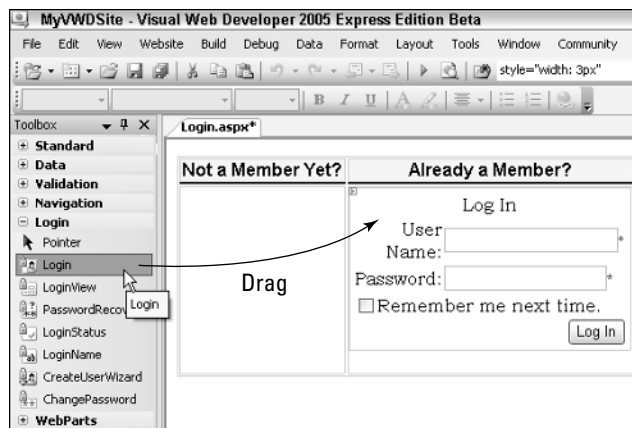
Adding a Server Control to a Page

Adding a server control to a page is pretty simple: You drag its name from the Toolbox onto your page. You can put server controls on a blank .aspx page, in Master Pages, or in the content placeholder in a page that uses a Master Page.

The size and complexity of server controls ranges from a tiny link on a page with a few properties to multipage wizards consisting of many smaller controls and many properties, and every size in between. Despite the differences among server controls, there are some commonalities in how you work with them. Those commonalities are what this chapter is mostly about.

As mentioned, adding a server control to a page is simple: You just drag its name from the Toolbox to wherever you want to place it on the page. In Figure 7-2, for example, I’ve created a page named `Login.aspx` and placed a table (along with some text) on the page. In the figure, I’ve just dragged a Login control from the Toolbox into the lower-right cell of the table.

Figure 7-2:
Using data
and .aspx
pages to
create
HTML.



Tweaking server controls in Design view

In Design view, clicking an ASP.NET control selects the control. The Properties sheet shows the name and properties of the control. The Properties sheet in Figure 7-3 shows an example in which a Login control is selected. The name of the control is `Login1`.

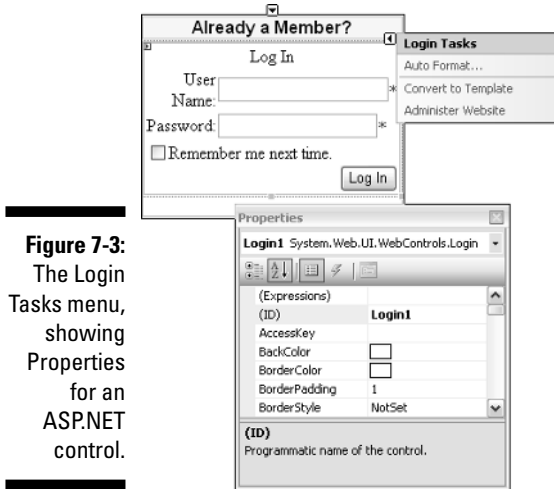


Figure 7-3:
The Login
Tasks menu,
showing
Properties
for an
ASP.NET
control.



Controls in the .NET Framework are organized into a hierarchical *namespace*, a place to keep them that works much the same way as the hierarchical organization of the folders on your hard disk that keep your files. In the Properties sheet of Figure 7-3, `System.Web.UI.WebControls.Login` is the full name of the Login control within the .NET Framework namespace.

You can design some elements of an ASP.NET control using the Style Builder. Just right-click the control and choose *Style*, or click the control so its name shows at the top of the Properties sheet. Then click the *Style* property and the *Build* button that appears. The Style Builder opens, and you can choose items as you would for any normal HTML control.

For example, to change the size of the text in the `Login1` control, scroll down its properties list and click the + sign next to the *Font* property. Then click the *Size* property and choose a relative size like *Medium*.



ASP.NET controls are converted to HTML before they're sent to the server. Element styles like `TABLE` and `TD` (for table cells) are applied to the HTML automatically. So don't knock yourself out trying to design an ASP.NET control in Design view until you've had a chance to see how it looks in a Web browser.

The Style Builder won't work for all parts of an ASP.NET control. If you want to fine-tune every single element in a control, you have to convert it to a template first. This is done on the Common Tasks menu you'll find on ASP.NET server controls. This menu is explored more fully in the next few sections.

Using the Common Tasks menu

In Design view, when you select (click) a server control, you see a tiny arrow button somewhere on its border. Clicking that tiny button opens the control's Common Tasks menu. In Figure 7-3, for example, I've clicked the little arrow button for a Login control on a page in Design view.

The exact options on a Common Tasks menu vary from one server control type to the next. But some options, such as

- ✓ Auto Format
- ✓ Convert to Template
- ✓ Administer Website
- ✓ Edit Templates

are so common they deserve further mention. The next few sections discuss these options in greater detail.

Using Auto Format

The easiest option on the Common Tasks menu is Auto Format. Clicking that button takes you to an Auto Format dialog box, where you see a list of scheme names and a preview window. When you click a scheme name, the preview shows you how the scheme looks.

If you find a scheme you like, just click OK to select it. Or, if you don't want to use a scheme and just want to use the default control appearance, click Remove Formatting, then click OK.

In Figure 7-4, I chose Auto Format → Classic scheme, and clicked OK. The Login control takes on the Classic scheme (it's blue on the screen, not gray). The change is also reflected in the control's BackColor, BorderColor, and other properties.



In the Properties sheet, values in boldface are recent. This serves as a good reminder that you can Undo them with Ctrl+Z or Edit → Undo.

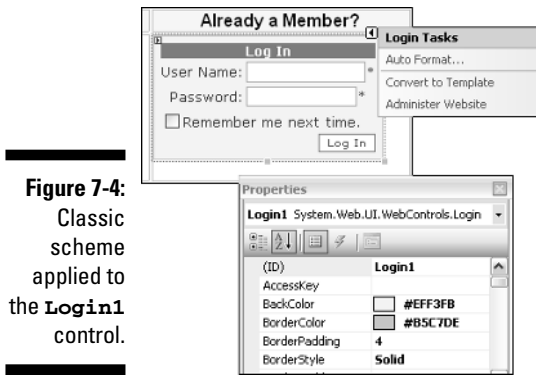


Figure 7-4:
Classic
scheme
applied to
the `Login1`
control.

Converting a control to a template

By looking at the Login control shown back in Figure 7-4, you can conclude that it must be made of several smaller controls. After all, it contains text, boxes, and a button, all neatly aligned. Where are those individual controls, and what if you want to change one?

The answer lies in converting the control from an individual control to a *template*. As its name implies, the template defines the exact content and position of each item that makes up the control.

When you choose Convert to Template, a complex visual control like Login is divided into its individual components. Then, the “Convert to Template” option on the Common Tasks menu is replaced with a Reset option.



The Reset option undoes changes you’ve made to the template before converting it back to a single control. Don’t click Reset if you intend to keep changes you’ve made in a template.

Within the template, each control has its own tiny arrow button — so tiny you practically need a microscope to see it. Fortunately, you don’t have to click the tiny ↔ button. It’s just there to identify the upper-left corner of each control in the template. Click anywhere on a control to select it. Figure 7-5 shows an example in which I’ve clicked the Log In button in the template.



To select multiple controls, click the first one, then hold down the Ctrl key as you click others.

The Properties sheet, as always, shows the name and properties of the selected control. Because it’s a server control, it won’t have the usual HTML tag name in angle brackets. You can change any property of the control in the usual manner. For example, to change the text that appears on the selected button, change the button’s `Text` property.

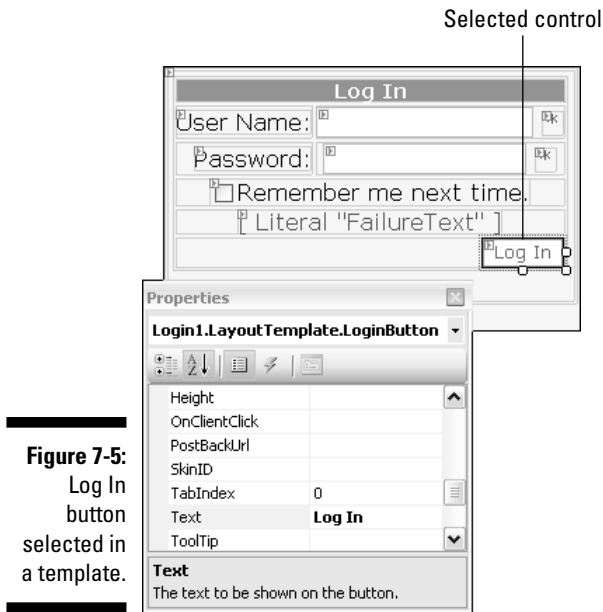


Figure 7-5:
Log In
button
selected in
a template.

Red text in a template represents text that appears in the browser only under special circumstances, such as when the user fails to fill in a required text box. You don't have to change any of the red text, it's just a temporary placeholder for normal text that will appear in the Web browser.

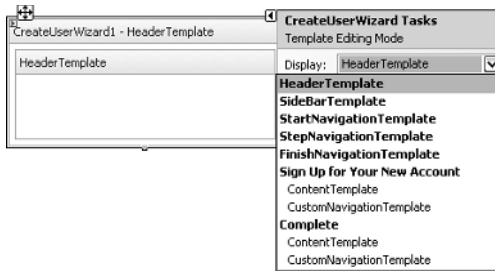
Administering the Web site

The Administer Website option on the Common Tasks menu is just a shortcut to the Web Site Administration Tool discussed in Chapter 3. There's really no need to switch to that tool, though. The link is only there as a convenience in case you want to check, or maybe change, current user accounts, roles, or access rules.

Editing templates

The Common Tasks menu of some server controls includes an Edit Templates option. For example, the multipage CreateUserWizard template described a little later in this chapter has an Edit Templates option on its Common Tasks menu. Clicking that option takes you to a template-editing window that has its own Common Tasks menu. From the Templates Common Tasks menu, you can choose which template (that is, which portion of the control) you want to style. Figure 7-6 shows an example where I chose Edit Templates from its common tasks menu after dragging a CreateUserWizard control onto a page.

Figure 7-6:
Common
Tasks
menu's
Template
Editing
Mode.



You can treat the template like a Web page in the sense that you can type text in the template or add controls to the template. If you're not sure where a template appears in a control, type in some random descriptive text. If you're editing a Header template, for example, type in "I am the Header Template." Then choose End Template Editing from the Common Tasks menu for the template. Afterwards, you'll see the control in its normal appearance, and the text you typed into the template appears within that control.

A big part of using server controls in a Web site is knowing how the control looks and acts in a Web browser. And to do that, use some controls in some pages. Because you went to great trouble in Chapter 4 of this book to configure your Web site to support member logins, I'll start by looking at some of the Login controls that you can use with that foundation.

ASP.NET Login Controls

Visual Web Developer supports several ASP.NET 2.0 controls that you can use to manage logins through your Web pages. They only work if you've already configured your site to support membership, as discussed in Chapter 4. You can use them in any Web Form (.aspx page). In the Toolbox for an .aspx page, you'll find all the Login controls under the Login heading, as in Figure 7-7.

Here's a quick overview of what each Login control is for:

- ✓ **Pointer:** Not really a Login control. If you click a control and then change your mind and want to get back to the normal mouse pointer, click this Pointer item.
- ✓ **Login:** Presents a control that allows users to log into their accounts with their user name and password.
- ✓ **LoginView:** Lets you show different stuff to different users based on whether they're anonymous or logged in.

- ✔ **PasswordRecovery:** As the name implies, presents fields that allow a user to recover a forgotten password.
- ✔ **LoginStatus:** Displays a Login link to anonymous users, or a Logout link to authenticated users.
- ✔ **LoginName:** Displays nothing to an anonymous user; displays an authenticated user's login name.
- ✔ **CreateUserWizard:** Provides a fill-in-the-blanks form for creating a new user account. Use it in your Web sites to allow people to create their own accounts.
- ✔ **ChangePassword:** Provides a form that allows a user to change his or her password.

You can use all of these controls anywhere in any `.aspx` page, any Master Page, as well as in the Content placeholder of a page that uses a master. The next few sections discuss these controls more fully. I'll start with the `CreateUserWizard` control.



Figure 7-7:
Login server
controls in
the Toolbox.

Allowing Users to Create an Account

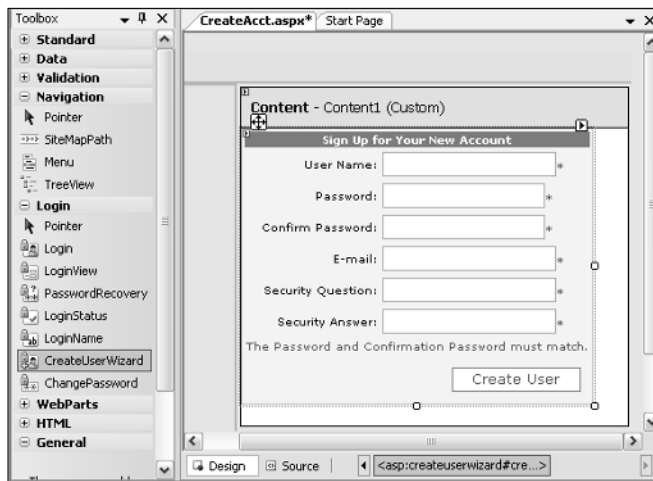
If your site will allow anyone to create an account, you need a page that allows users to enter the appropriate data. The `CreateUserWizard` control is just the ticket to do this because it displays on a Web page all the fields needed for a user to create an account.

To create a new page for the control, right-click the site folder or subfolder in Solution Explorer and choose Add New Item. Choose Web Form and fill in the blanks as you see fit. I named mine `CreateAcct.aspx`. The other options are entirely up to you. I chose Visual C# as the language, as I routinely have in the other examples in this book, and selected the Place Code In Separate File option. I also used the Master Page described in Chapter 4 in my example.

Anyway, after you've created (or opened) a page, just drag a `CreateUserWizard` tool from the Login category of the Toolbox onto the page (or drag it into the Content placeholder in a Master Page). If you like, use the Auto Format option on the Common Tasks menu to style the form.

Figure 7-8 shows an example in which I've added a `CreateUserWizard` control to the Content placeholder on a page. I also used Auto Format on the Common Tasks menu to set the scheme to Classic, making the control easier to see.

Figure 7-8:
Create
User
Wizard
control in
Design
view.



If you want to keep life simple, just close and save the page. You don't really have to do anything else to the control, all the stuff it needs to validate the user's entries, store the user's data in the SQL Server database, and so forth, is already done.

If you want to customize the control later, you can do so at any time. It has a Common Tasks menu, templates, and so forth. You can press Help (F1) for help while customizing the control.

Assigning new users to a role

The `CreateUserWizard` control doesn't automatically assign a new user to a role. If you want to put all the people who sign up through the `CreateUserWizard` control into a specific role, add a little code. I know I haven't talked about code in this book, and now is not the time to get into details of writing code. However, for this example, I can just show you what to type.



This technique only works if you've already enabled Roles in the Web Site Administration Tool as discussed in Chapter 3. Also, you'll have to remember exactly how you spelled your role name, as it won't appear in any IntelliSense menu.

If you want your `CreateUserWizard` control to automatically assign every new user to a specific role in your site, follow these steps:

- 1. In design view, select the `CreateUserWizard` control.**

Its name appears in the Properties sheet when selected.

- 2. In the Properties sheet, click the Events (lightning bolt) button.**

- 3. Double-click the `CreatedUser` event.**

You're taken to the code-behind page for the page you're designing; the cursor is already in the event procedure, right where you type its code.

- 4. Type the following at the cursor position, substituting your own data for the italics, as follows:**

```
Roles.AddUserToRole(ctrlName.UserName, "roleName");
```

ctrlName refers to the name of your `CreateUserWizard` control. That would likely be `CreateUserWizard1` if you let VWD name the control. When you type a *roleName*, make sure you spell it exactly as you did when creating the role in the Web Site Administration Tool.

The code given here shows what my example looked like after adding a line of code to add the new user to my `SiteMembers` role.

```
protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
{
    Roles.AddUserToRole(CreateUserWizard1.UserName, "SiteMembers");
}
```



Typing code isn't like typing in English. In code you have to get every dot, comma, parenthesis, quotation mark, and semicolon exactly right. Spelling and blank spaces count a lot too. The code won't work if you don't type the line exactly right.

- 5. Close and save the code-behind page.**

A couple' CreateUserWizard tips

If you take a look at the whole Properties sheet for the `CreateUserWizard` control, you'll see it has many properties. You can figure out what most properties are just by looking at their names and values.

A couple of noteworthy properties include `ContinueDestinationPageURL` and `LoginCreatedUser`. You can set the `Continue`

`DestinationPageURL` property to whatever page you want to have open when the user clicks `Continue` after successfully creating an account. The `LoginCreatedUser` property determines whether or not the user is logged in automatically after creating the account. The default is `True`, the user is logged in.

Testing the control

After you close and save the new page, you can take the `CreateUserWizard` control for a test drive. Just right-click the page's name (`CreateAcct.aspx` in my example) in Solution Explorer, and choose `View in Browser`. When the page opens, you should be able to create a new user account. You'll have to remember the user name and password to test the account later.

Figure 7-9 shows an example of a user account (in the Web browser) with the user name `Testy`. The hypothetical data you enter needs to be realistic enough to pass all the tests that the control imposes. For example, if the passwords don't match, you'll get an error message in the control and won't be able to create the account.



Don't forget that your password needs to be at least seven characters long, and must include a non-alphanumeric character. For example, `password!` (with the exclamation point) is an acceptable (if rather obvious) choice.

Figure 7-9:
Create User Wizard control in the Web browser.

Sign Up for Your New Account	
User Name:	<input type="text" value="testy"/>
Password:	<input type="password" value="•••••••"/>
Confirm Password:	<input type="password" value="•••••••"/>
E-mail:	<input type="text" value="trash@coolnerds.com"/>
Security Question:	<input type="text" value="Favorite insect"/>
Security Answer:	<input type="text" value="mantis"/>
<input type="button" value="Create User"/>	

When the control contains reasonable data for a hypothetical user account, click the `Create User` button. You should see a "Your account has been successfully created" message and a `Continue` button. At this point there's no place to continue to, so you can just close the Web browser to return to VWD.

To verify that the process worked, you can choose Website ⇄ ASP.NET Configuration from the menu bar to open the Web Site Administration Tool. Click the Security tab, and then click Manage Users. The new user should be listed. If you added code to add the user to a role, click the [Edit Roles](#) link for that user. The check box should be filled already, indicating that the user is in the role.

When you have a way for users to create an account on your site, the next thing they need is a way to log in with their user names and passwords. That's where the Login control comes into play.

Creating a Login Page

People who have already created accounts at your site need a place to sign in. Some of the other Login controls assume that page is in the site's root folder and named `Login.aspx`. So if you haven't already done so, you should create that page now by following the usual steps:

- 1. In Solution Explorer, right-click the site folder at the top of the folder hierarchy and choose Add New Item.**
- 2. In the Add New Item dialog box, choose Web Form.**
- 3. Name the page `Login.aspx`.**
- 4. Choose C# as the language and choose Place Code In Separate File.**

Whether or not you choose to use a Master Page is entirely up to you.

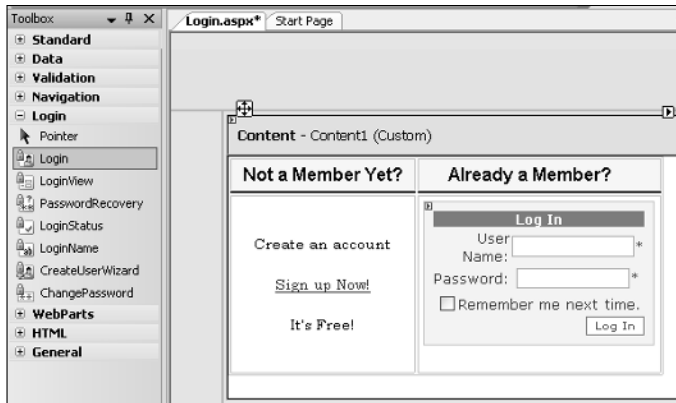
- 5. Click the Add button.**

If you opted to use a Master Page, click the Master Page's folder, and then click the Master Page filename and click OK.

I created my Login page using a table with two rows and two columns, and typed in a couple column headings as described in Chapter 5. I dragged the Login control from the Toolbox into the lower-right cell. I used its Common Tasks menu to apply the Classic scheme. Figure 7-10 shows the Login tool (highlighted in the Toolbox) and a Login control placed under the "Already a Member?" heading in a table.

In the cell to the left of the Login control, I typed the Sign Up Now! text and selected it. Then I used the Convert to Hyperlink button in the toolbar to browse to the `CreateAcct.aspx` page I created previously. (In the Hyperlink dialog box, the Type appears as (other) and the URL as the name of the page.) Click OK, and the page offers non-members a way to create an account right now.

Figure 7-10:
Login
control
added to
a page.



To align text in a table cell, right-click the cell and choose **Style**. Click **Text** in the Style Builder, set the Horizontal and Vertical Alignment, and click **OK**. To widen a column beyond what you can do by dragging, right-click some empty space in a cell in the column and choose **Resize** ⇄ **Resize Column**. Then, increase the current width, choose **All Cells**, and click **OK**.

Like all ASP.NET controls, the Login control has a Common Tasks menu, as you saw earlier in this chapter. However, everything there is optional. All you really have to do is close and save the page on which you placed the control.

To test the login page, right-click its name `Login.aspx` in Solution Explorer and choose **View in Browser**. To verify that the control works, type in a valid user name and password. An invalid user name and password is rejected. A valid entry takes you to the default page for the site. But that's fine for now; there's no way to verify right now if you're logged in or not anyway. This brings me to some other Login controls. (Don't forget to close the Web browser.)

Providing a Login Link

There's no telling which page in your site a user might first encounter. A link from a search engine could take a user to any page. To make your site easy to use, you need a login link on every page. An easy way to provide this is to put a link to the `Login.aspx` page into the Master Page for your site.

But a regular link that always showed Login would be confusing for people who have already logged in. The link should show Logout or something else for people who are already logged in. That's where the `LoginStatus` control comes in.



You must create a page named `Login.aspx` in your site's root folder before you can actually use the controls described in this section. Otherwise, when you try to test the control in a Web browser, you'll get an error message indicating that the `Login.aspx` page cannot be found.

The *LoginStatus* control

The easiest way to provide a Login/Logout link is to drag a `LoginStatus` control to a Master Page. That's all you really have to do, besides close and save the Master Page.

Then, right-click any page that uses the Master Page and choose View in Browser. In the browser, clicking the Login link should take you to `Login.aspx`. If you enter a valid user name and password, as in Figure 7-11, and click Log In, you'll be logged in. If your `Default.aspx` page uses the Master Page, you'll see that the link has changed from Login to Logout, because you're logged in.

Figure 7-11:
Putting
Login.aspx
to the test.

Not a Member Yet?	Already a Member?
Create an account Sign up Now! It's Free!	<div style="border: 1px solid gray; padding: 5px;"> <div style="background-color: #cccccc; text-align: center; padding: 2px;">Log In</div> User Name: <input type="text" value="Testy"/> Password: <input type="password" value="*****"/> <input type="checkbox"/> Remember me next time. <div style="text-align: right;"><input type="button" value="Log In"/></div> </div>

If you see a Logout link, click it to log out. You won't be taken to another page. You'll simply be logged out and the control will again show Login. Close the Web browser. If all of that works, you have all the basic stuff for a membership site: a way to sign up for an account, a way to log in, and site-wide Login and Logout links. Other stuff to follow in this section is optional stylistic stuff.

By default, Logged Out users see a Login link, and authenticated users see a Logout link. In Design view, the `LoginStatus` control has all the usual design accoutrements, including a Properties sheet and Common Tasks menu (Figure 7-12). You can change the text of either link using the `LoginText` and `LogoutText` properties.

As an alternative to using text links, you can choose a pair of graphic images to show. One image to show to anonymous users, and another to show to authenticated users. When those pictures are placed in your site's folders, use the `LoginImageUrl` property of the control to specify the picture to show to anonymous users. Set the `LogoutImageUrl` to the picture that authenticated users should see.

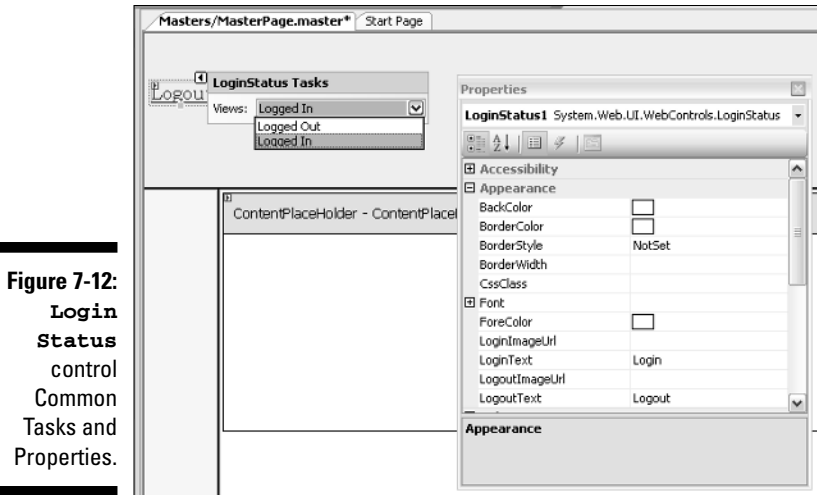


Figure 7-12:
Login
Status
control
Common
Tasks and
Properties.

The Views menu on the `LoginStatus` control's Common Tasks menu lets you switch between what anonymous (Logged Out) users will see, and what authenticated (Logged In) users will see. You can change the text of either link just by editing the text.

If you want something a little fancier than what the `LoginStatus` control has to offer, consider the `LoginName` and `LoginView` controls, discussed in the next two sections.

The LoginName control

The `LoginName` control is about as easy to use as a control can be. It shows nothing to anonymous users. For authenticated users, the control displays the user's login name. Typically you use the `LoginName` control with the `LoginView` control.

The LoginView control

Like the `LoginStatus` control, the `LoginView` control can tell the difference between anonymous and authenticated users. But it's not limited to showing text or a picture. You can use it to show just about anything, even ASP.NET server controls. To use the control, just drag it to the Web page or Master Page.

After the control is placed on the page, the control is shaped as a box. The box has two different views, but you only see one view at a time. You can choose which view you want to see (and design) by choosing one of the following from the View option on the `LoginView` control's Common Tasks menu:

- ✓ **Anonymous Template:** Content that anonymous users see.
- ✓ **LoggedIn Template:** Content that only authenticated users see.

What you put into either template is entirely up to you. It can be text, a table, a picture, an ASP.NET control, whatever. Think of the box as a mini-page that can contain anything that a big page can contain.

During this stage of the site-building project, your best bet would be to create a simple `LoginView` control that shows a Login link to anonymous users. For authenticated users, the control shows the user's name and a Logout link. This comes in very handy when you're testing your site, because you can always see whether you're currently testing as an anonymous user or as an authenticated user. Putting the control on a Master Page is especially helpful because you'll be able to see it on every page that uses the Master.

So, given that general advice, let's take a look at the steps required to make it happen:

1. In Solution Explorer, double-click your Master Page to open it in Design view.

In Chapter 4, I created a `MasterPage.master` file in a folder named `Masters`. So I'll use that one as a working example here.

2. Drag a `LoginView` control from the Login category of the Toolbox to the top (or left) pane of the Master Page so that it will be visible on all pages that use the master.

By default, the control will be named `LoginView1`.

3. From the control's Common Tasks menu, choose Anonymous Template.

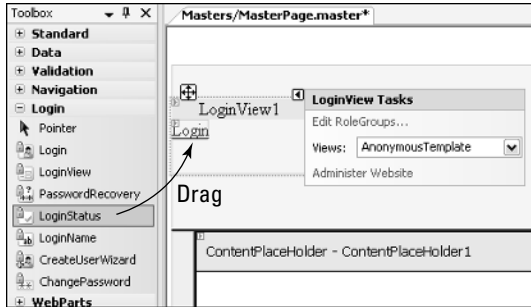
Whatever you add to the `LoginView` control now will be visible only to anonymous users.

4. Drag a `LoginStatus` control from the Toolbox into the `LoginView1` control.

The `LoginStatus` control appears as a Login link, and its Common Tasks menu might open automatically. You don't need to change anything on the `LoginStatus` control, so just click the larger `LoginView1` control to hide that Common Tasks menu.

So at this point, the Anonymous Template for the `LoginView` control contains a Login link. If you click the Common Tasks button for the `LoginView` control, you should see `AnonymousTemplate` in the Common Tasks menu, and the Login link inside the `LoginView1` control as shown in Figure 7-13.

Figure 7-13:
Login
View
control's
Anonymous
control
contains a
Login
Status
control.



5. To choose what authenticated users will see in the control, click the Common Tasks button for the `LoginView1` control and choose `LoggedInTemplate`.

The Login link disappears because it's on the Anonymous Template, not the LoggedIn Template.

6. In the `LoginView1` control, type the word Hello followed by a blank space.
7. Drag a `LoginName` control from the Toolbox into the `LoginView1` control, just to the right of the word Hello.
8. Drag a `LoginStatus` control from the Toolbox into the `LoginView1` control.

At this point, the `LoggedInTemplate` should look like Figure 7-14. When you view the Common Tasks menu for the `LoginView1` control, it should show `LoggedInTemplate`, as shown in the figure.



The `LoginStatus` control always shows the word Login in Design view. Don't worry about that. Later, when you actually use the page in a Web browser, it will show Logout to authenticated users.

You can't open a Master Page on its own in a Web browser, so you can't choose View in Browser to test the page immediately. But that is not a problem. You'll be able to put things to the test a little later in this chapter. For now, just close and save the Master Page.

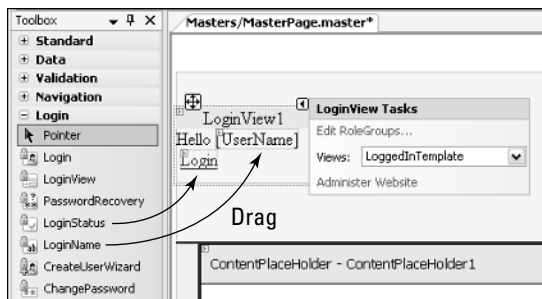
Multiple views for multiple roles

The `LoginView` control isn't limited to showing two different views. If your site contains numerous roles, you can use the `Edit Role Groups` option on the `Common Tasks` menu to define multiple views for multiple roles. In the `RoleGroup` Collection Editor, use the `Add` button to add a `RoleGroup` placeholder to the left column. Then, to the right of the `Roles` column in the right column type a valid role name from

your site, and then click `OK`. You can repeat the process to add multiple role groups.

After you've created a `RoleGroup`, you can design what it shows to its members by clicking the `Views` option on the `Common Tasks` menu and choosing the new `RoleGroup` name. In other words, whatever you put in the `LoginView` control at this point is visible only to people in the specified role.

Figure 7-14: LoggedIn Template contains text and Login Name and Login Status controls.



Letting Users Manage Passwords

People forget their passwords all the time. And the last thing you need is to be spending all your time reminding people of their forgotten passwords. So your site needs a means of allowing users to retrieve their own passwords. Likewise, users have to be able to change their own passwords, so your site will need that capability too. In the `Login` category of the `Toolbox`, the `PasswordRecovery` and `ChangePassword` controls are just the ticket.

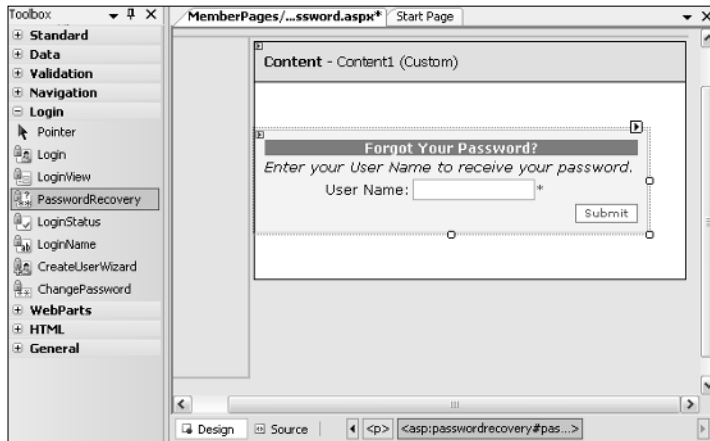
Using the PasswordRecovery control

The `PasswordRecovery` control provides a way for a user to retrieve a forgotten password. (Actually, with the default hashed encryption used in Visual Web Developer, it sends them a new password that they can use to log on). In

the browser, the control first asks the user to enter a user name and click Submit. If the user name exists in the database, the second page appears showing the user's secret question. When the user enters the correct answer and clicks Submit, the control e-mails a password to the user's e-mail address, and provides a "Your password has been sent to you" confirmation message.

To use the control, create a new page or open the page on which you want to place the control. For my example I created a new page named `Recover Password.aspx` that uses my Master Page. Just drag the control onto the page, like any other server control. You can choose a color scheme from the Auto Format option on the control's Common Tasks menu. I applied the Classic scheme to the `PasswordRecovery` control shown in Figure 7-15.

Figure 7-15:
The
Password
Recovery
control.



The `PasswordRecovery` tool is unique in that there are a couple of extra steps involved in getting it to work. Furthermore, the page has to be configured to work on the Web server, not your local PC, so you may need to just leave that page as-is, and then remember to finish it later after you've copied the site to a Web server.

One thing you need to configure is the return e-mail address. This will likely be an e-mail address you create using your own domain name (after you have set up your own domain name). You enter that return mailing address into the Properties sheet by following these steps:

1. With the page open and visible in Design view, click the `PasswordRecovery` control to select it.
2. In the Properties sheet, expand the Behavior and MailDefinition categories.

3. Type your return e-mail address as the **From** property.

Figure 7-16 shows an example where I've entered the hypothetical address `support@yourdomain.com`.

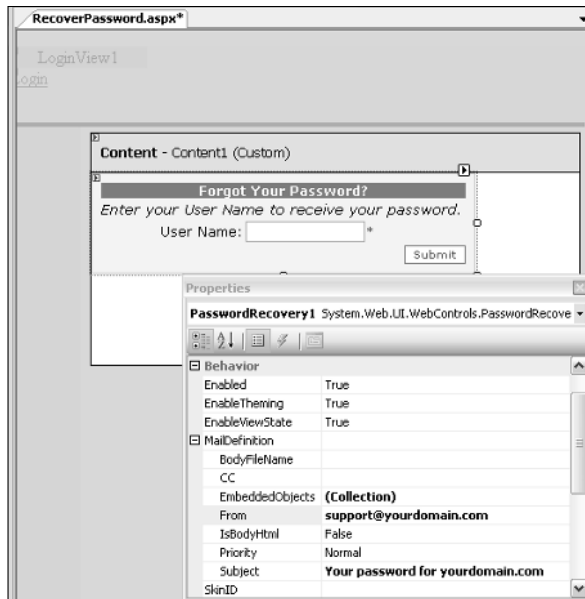


Figure 7-16:
Set the **From** property to an address for your own domain.

4. Optionally, you can also fill in the **Subject** line for the message.

In addition to defining a return address, your site must be configured to use the SMTP (Simple Mail Transfer Protocol) server provided by your hosting service. You won't know what that is until you've set up an account with a hosting provider, so don't knock yourself trying to get the `PasswordRecovery` control to work on your own PC. The `PasswordRecovery` control doesn't even need to work on your PC: It only has to work on the Web server.

So the only smart thing to do is close and save the page that contains the `PasswordRecovery` control, and forget about it for now. If you try to test it in a Web browser, you'll just get an error message when it tries to e-mail the password. When you've chosen a hosting provider, they will tell you how to configure your site to use their SMTP mail server. It *may* be something as simple as adding the following lines to your `Web.config` file:

```
<smtpMail
  serverName="localhost">
</smtpMail>
```

Testing RecoverPassword on your Local PC

I don't mean to imply that it's absolutely impossible to test the `RecoverPassword.aspx` page on your local PC. The fact is, if you know how to configure IIS, understand virtual directories, and have access to an e-mail server, you could get the control to work. However, doing so doesn't really solve anything because getting

the control to work on your local PC is irrelevant. The control only needs to work on the Web server. Even if you did get the control to work on your local PC, you'd probably still have to reconfigure on the Web server to get it to work in your actual, live Web site.

But keep in mind that I said it *may* be something as simple as that. Only your hosting provider can tell you specifically what's needed to get your `PasswordRecovery` control to work on their Web servers. Furthermore, and this is important enough to put in a warning:



If you add the `<smtpMail>` tag shown above to the `Web.config` file on your local PC, you might create a situation where your site doesn't work at all. Wait until you've actually posted your site to a Web server to configure SMTP mail for your site.

Even though you can't put the `RecoverPassword.aspx` page to the test yet, you still need to provide a link to the page so that once the page is published, people can get to the page. And ideal place for the link would be the `Login.aspx` page, where users will likely first discover they've forgotten their password. Figure 7-17 shows an example where I've added a link to the `Login.aspx` page that asks if the user has forgotten his password. The target of that link is the `RecoverPassword.aspx` page.

Figure 7-17:
"Forgot your password?"
This link was added to `Login.aspx`.



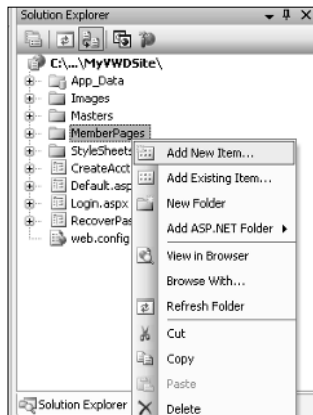
The ChangePassword control

As its name implies, the `ChangePassword` control lets a logged-in user change her password. Unlike other pages you've created in this chapter, a "Change Password" page applies only to logged-in users. An anonymous user can *create* an account and password, using the `CreateAcct.aspx` page described earlier. An anonymous user can also *recover* a lost password, assuming he or she has set up an account in the past and has simply forgotten their password. That user would access the `RecoverPassword.aspx` page to get their password. But again, you won't be able to really put `RecoverPassword.aspx` to the test until after the site is on a Web server.

Because only logged-in users can change their password, you can put the page that allows password changes into the protected `MemberPages` folders. The other pages described in this chapter, `CreateAcct.aspx`, `Login.aspx`, and `RecoverPassword.aspx`, need to be in the root folder, or some other folder to which anonymous users have access.

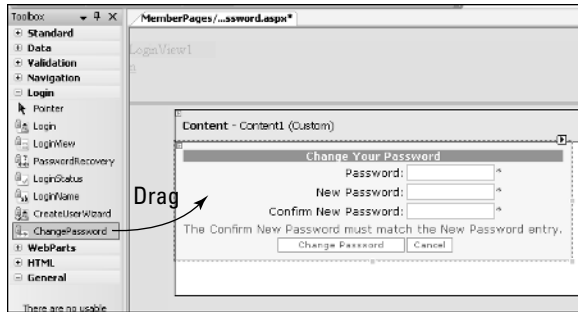
To put the `ChangePassword.aspx` page in the `MemberPages` folder, right-click that folder name in Solution Explorer and choose `Add New Item` as shown in Figure 7-18. In the `Add New Item` dialog box that opens, be sure to choose `Web Form`. I named my page `ChangePassword.aspx`, but you can name yours as you see fit.

Figure 7-18:
About to
add a new
page to the
Member
Pages
folder.



As with any page, you can add text, tables, pictures, or whatever to make it look however you like. The only control you must add to the page is a `ChangePassword` control. Just drag that control name from the `Login` category of the `Toolbox` onto the page, as shown in Figure 7-19.

Figure 7-19:
Change
Password
control
added to
a page.



In the figure, I applied the Classic scheme to the `ChangePassword` control, just to give it some color. But even that step is optional. The control doesn't require any further configuration, so you can just close and save the page.

Testing Membership

At this point, you have built some more of your site's membership infrastructure. When you test pages by viewing them in a Web browser, you will be experiencing the site exactly as strangers who are visiting the site will experience it. That can be very confusing if you forget about the access rules you defined back when you were first configuring membership.

For example, if you right-click the `ChangePassword.aspx` page and choose View in Browser, you might be shocked to discover that the `ChangePassword.aspx` page doesn't open. Instead, the `Login.aspx` page opens! Most confusing indeed! But it's not an error or a problem. It's the way things are supposed to work. Here's why.

Any page that's stored in the `MemberPages` folder is off-limits to anonymous users (assuming you created an access rule to make it off-limits, as described in Chapter 3). When an anonymous user attempts to open a page in the `Member Pages` folder, she is automatically redirected to the `Login.aspx` page. If you're not signed into an account when you try to open `ChangePassword.aspx`, the same rule applies to you. Opening `ChangePassword.aspx` automatically redirects you to `Login.aspx`.

If you sign into a user account on the `Login.aspx` page and click the Submit button, *then* you'll be taken to the `ChangePassword.aspx` page. It's important to understand how that works, otherwise you'll drive yourself absolutely batty trying to open members-only pages from the standpoint of an anonymous user, because every page you try to open will take you to `Login.aspx` until you actually log into a user account.

The new `LoginView` control at the top of the Master Page is a big help in that regard. Because any time you open in a Web browser a page that uses that Master, you'll see your current status right away. If you're currently testing things out as an anonymous user, you'll see a `Login` link in the Master, as shown at the top of Figure 7-20. When you're testing things out as a logged-in user, you'll see `Hello`, your current user name, and a `Logout` link, as shown at the bottom side of that same figure.

Figure 7-20:
Login
View
control as
seen by
anonymous
user
(top) and
authenti-
cated users
(bottom).



Keep in mind, too, that any user accounts that you create through the `CreateUserWizard` control are actual user accounts that will be stored in the database. You can view all current user accounts at any time via the Web Site Administration tool. Here's how:

1. From Visual Web Developer's menu bar, choose **Website** ⇨ **ASP.NET Configuration**.
2. In the Web Site Administration tool, click the **Security** tab.
3. Click **Manage Users**.
4. To see what role any user is in, click the **Edit Roles** link in that user's role.

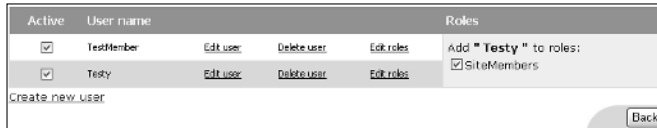
For example, in Figure 7-21 I created a new user account for a hypothetical user named Testy. Clicking the `Edit Roles` link for that user shows that the user has indeed been added to the `SiteMembers` role, as per the code added to the page to ensure that each new user is assigned to the `SiteMembers` role.

Because there's no way to recover a forgotten password on your local PC, if you forget the password for any sample user account you create, your best bet would be to just delete the account by clicking the `Delete User` link next to the account name. Then you can re-create the account with a password you'll

remember. You can create the account using either the Web Site Administration Tool, or your `CreateAcct.aspx` page. It doesn't matter which you use because the result will be the same: The user is added to the database.

Figure 7-21:

Testy is added to the Site Members role.



Server Controls in Source View

When you drag a server control from the Toolbox to the page, you add a pair of `<asp:>...</asp:>` tags that define that control to the Source of your page. You can see for yourself by clicking the Source button after adding a server control to a page. For example, the tags representing a Login control would look something like this:

```
<asp:Login ID="Login1" runat="server">
</asp:Login>
```

Most of the server controls you create should follow the same general syntax, as summarized here:

```
<asp:controlType ID="yourName" runat="server">
</asp:controlType>
```

where

- ✓ *controlType* is the *type* of control, and matches the name shown in the Toolbox.
- ✓ *yourName* is the name that uniquely identifies the control. VWD creates a default name, such as `Login1`. You can replace the default name with a name of your own choosing.
- ✓ `runat="server"` identifies the item as a server control, to be executed on the Web server rather than on the client computer.

The tags for a control really don't look like much, especially if you just keep the default settings for the control. In fact, from looking at the tags in Source view, you'd wonder how they accomplish anything at all — there's really not much to them.

But, as is often the case, first appearances don't tell all. There are actually lots of attributes and settings hidden inside the `<asp> . . . </asp>` tags. These attributes are just intentionally hidden so as to avoid cluttering up the Source view of the page.

If you switch back to the Design view, and use the control's Common Tasks menu to convert the control to a template, the `<asp> . . . </asp>` tags in Source view will change, often dramatically. Even if converting a control to a template has absolutely no visible effect on the control in Design view, chances are the switch has had a big effect on the content of the `<asp> . . . </asp>` tags.

For example, when you convert a Login control to a template, and switch to Source view, the number of tags between the Login control's opening and closing `asp` tags increases — dramatically. In fact, I can't show all the tags here — they'd take several pages to display.

Included in the expanded template view of the server control are the actual HTML tags used to render the control in the user's Web browser. You can edit any attribute in any tag you like (you can even design your controls that way).



In Design view, choosing Reset from the Common Tasks menu collapses the control back to its smaller size in Source view. But remember, the Reset option also cancels out any customization you did while in the template view.

I imagine most people would find it tedious to design things by tinkering with individual attributes in a templated server control. But then again, it all depends on your background and experience. There's no rule that says you *must* work in Source view. But you can if you want to — do what works.

Relaxing Password Constraints

By default, Visual Web Developer requires that passwords be at least seven characters long and contain at least one non-alphanumeric character. This provides for strong security — perhaps stronger than your site really needs. If you're not storing personal or financial information about users, you may want to relax the rules a little so users can make up passwords that are easier for them to remember.

To relax the password rules, you need to edit the `Web.config` file in your site's root folder. To edit the `Web.config` file, just double-click its name at (or near) the bottom of Solution Explorer. Initially the file contains some XML tags that look something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms" />
  </system.web>
</configuration>
```

You need to add some new tags above the closing `</system.web>` tag. You must type carefully because even the slightest mistake will prevent your site from working. (In fact, I'll post the exact text in the Chapter 7 section of my Web site at www.coolnerds.com/vwd so you can copy-and-paste rather than type). The exact lines to add above the `</system.web>` tag are as follows:

```
  <membership>
    <providers>
      <remove name="AspNetSqlMembershipProvider" />
      <add name="AspNetSqlMembershipProvider"
        type="System.Web.Security.SqlMembershipProvider"
        connectionStringName="LocalSqlServer"
        minRequiredPasswordLength="5"
        minRequiredNonalphanumericCharacters="0"
        passwordStrengthRegularExpression=""
      />
    </providers>
  </membership>
```

The `minRequiredPasswordLength` setting determines the minimum number of characters required for a password to be valid. I set that to five characters in the example. You can use a different minimum length if you prefer.

The `minRequiredNonalphanumericCharacters` setting determines how many non-alphanumeric characters (punctuation marks) are required. I set that to zero so no punctuation marks are required.

The `passwordStrengthRegularExpression` setting defines a *regular expression* that could require that the password contain (or not contain) certain characters. By setting that to nothing (""), as in the example, you allow "normal" passwords that can contain only text.

Figure 7-22 shows how the `Web.config` file should look after adding the appropriate lines to the file. Notice how the new lines are all together within the `<system.web>` and `</system.web>` tags.

Figure 7-22:
Web.config
file with
code to
relax
password
restrictions.



```
web.config
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms" />

    <membership>
      <providers>
        <remove name="AspNetSqlMembershipProvider"/>
        <add name="AspNetSqlMembershipProvider"
            type="System.Web.Security.SqlMembershipProvider"
            connectionStringName="LocalSqlServer"
            minRequiredPasswordLength="5"
            minRequiredNonalphanumericCharacters="0"
            passwordStrengthRegularExpression=""
            />
      </providers>
    </membership>
  </system.web>
</configuration>
```

Make sure you close and save the `Web.config` file after making any changes. To test the change, view the `CreateAcct.aspx` page (described earlier in this chapter) in a browser. (If you get an error message rather than the page, you typed something wrong in the `Web.config` file). You should be able to add a new test user account using a five-character (or more) password that contains no punctuation marks.

Chapter 8

Easy Site Navigation

In This Chapter

- ▶ Getting organized, staying organized
 - ▶ Creating custom site-navigation controls
 - ▶ Using Menu, TreeView, and SiteMenuPath controls
 - ▶ Make life easier with Web User Controls
-

If your site is to be successful, it must be easy to navigate. If people can't easily find and get to what they need, they might quickly lose interest and move on to another site. If yours is a large Web site, navigating between pages using nothing but hyperlinks can be tedious for the user. But for you, the developer, managing a large site with too many links can also be a nightmare.

Visual Web Developer's Site Navigation controls provide a great way to provide an easy, consistent navigational structure for your site. Furthermore, you can define all the links in one place. That way, when you add a new page to the site (or delete a page), you don't have to worry about going back to tweak the links in a bunch of different pages in your site — you just have to keep the links straight in one file. This chapter gives you the goods on how to manage your sites with Site Navigation controls.

Getting Organized

At some point, you have to think about how you're going to organize your Web site's content. Exactly how you do that depends on your content, the size of your site, and your definition of the word *organized*. But if you intend for your site to have major areas that people can navigate to, it would be good to create a file for each of those major areas. You can even put a blank `.aspx` page in each one as a placeholder to link to when you're developing your site navigation.

In Figure 8-1, for example, I've created some folders within my PublicPages folder. I've put a blank .aspx page in each folder. You don't have to include all your site's folders in your navigation structure. In the interest of keeping this example to a reasonable size, I'll just focus on the AboutUs, Help, Products, and Services folders in my PublicPages folder.

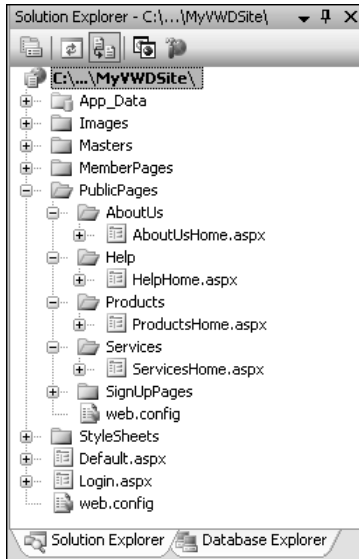


Figure 8-1:
Some
folders and
pages in the
PublicPages
folder.

Using Site-Navigation Controls

The two main VWD controls for site navigation are `Menu` and `TreeView`. The `Menu` control offers a simple drop-down menu of navigational links, as shown at the left of Figure 8-2. When the `Menu` control is first displayed on the page in a browser, only the Home link and arrow symbol are visible. The menu of links doesn't appear until the user clicks the arrow button. The `Menu` control is good for a small menu that you want to keep out of the way most of the time.



Figure 8-2:
Menu
(left) and
TreeView
(right)
examples.

The `TreeView` control shows the navigational structure in a collapsible tree, as in the example at right in Figure 8-2. The user can click + and – signs to (respectively) expand and collapse categories. I’m sure you’ve seen a gazillion similar collapsible trees; in VWD, both the Toolbox and Solution Explorer are collapsible trees. The `TreeView` control is best for handling larger navigational tasks.

Both the `Menu` and `TreeView` commands can be used with either *static data* or *dynamic data*. When using a control with static data, the navigational structure of the site is defined as part of the control. The method is easier in that you can define the whole menu structure just by filling in the blanks in dialog boxes.

With dynamic data, you store data about the site’s structure in a file, called a *site map*, that’s external to the control. The advantage to this approach is that the site’s navigational structure is stored in one place. So if you need to change the navigational structure of the site, you just have to change the external file, not every control on every page.

In the sections that follow, I’ll look at using both controls with static data and with dynamic, external data.

Using the TreeView and Menu Controls

The `TreeView` and `Menu` server controls are both in the Navigation category of the Toolbox. They’re so similar that reading one set of instructions is sufficient for you to use either. In this section, I describe how to use the controls with static data (without a site map that’s defined in an external file). This is the easiest way to create a map, especially if it’s a small map that’s not likely to change often. (That’s especially true if, like a lot of us, you know nothing about XML and can’t type worth beans.)

The only real drawback to this method is that it’s tedious. And if you put the control on a specific Web page, you’d have to re-create the control on (or copy and paste the control to) other pages. (Unless you put the control in a Master Page, in which case it will appear on all pages that use the Master Page — which may not be what you’re looking for.)

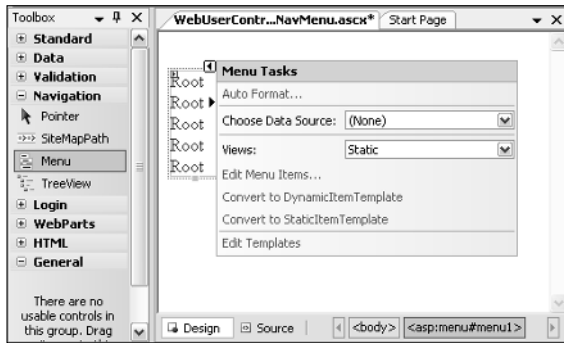


If you don’t want to put the navigation control on a Master Page, you could put it in a Web User Control and use it as needed on pages throughout your site. I’ll talk about Web User Controls later in this chapter.

So the first step is to open the page on which you want to place the `Menu` or `TreeView` control. Make sure you’re in Design view. Then follow these steps:

1. Drag either a **Menu** or **TreeView** control from the **Toolbox** onto the page as shown in **Figure 8-3**.
2. On the **Common Tasks** menu, click **Edit Menu Items**.
The **Menu Item Editor** dialog box opens.
3. Use the **Menu Item Editor** to define each menu item.
More on this rather hefty step in a moment.
4. Click **OK**.

Figure 8-3:
Menu
control
added to
the page.



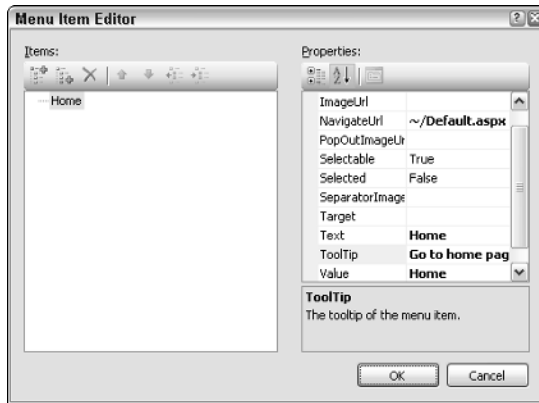
To use the **Menu Item Editor**, use toolbar buttons at the top of the **Items** list to insert options to appear on your menu. You'll need a root element first, so click the **Add Root Item** button in the toolbar to add a new root. At first, the item is just named **New Item**.

Next you have to set the following properties in the **Properties** column of the dialog box:

- ✓ **NavigateURL:** Click this property, and then click the **Build** button that appears. Use the **Select URL** dialog box that opens to navigate to the page that the link should open, and then click **OK**.
- ✓ **Text:** Type the text of the menu item.
- ✓ **ToolTip:** Type the **ToolTip** text for the item.

Figure 8-4 shows an example where I've added one root element. I've set its **NavigateURL** property to `~/Default.aspx`. The `~` character always refers to the root folder of the site (C:\...\MyVWDSite in **Figure 8-1**). So `~/Default.aspx` means "the `Default.aspx` page in the site's root folder." I changed the **Text** property to **Home** (which also changed the text of the **Value** property). And I added a little **ToolTip** that reads `Go to home page`.

Figure 8-4:
One item
added to
Menu Item
Editor.



To add a child page item under the root item, click the Home item at the top of the left column, and then click the Add a Child Item button in the toolbar (second button from the left). Another New Item appears, indented under the first item.

As before, you want to set the `NavigateUrl` property to the page the menu item opens, the `Text` to the item as you want it to appear in the menu, and optionally a `ToolTip`. Just keep repeating the process until you've defined all your items.



Use other buttons in the toolbar to work with items you've already put in the menu. Click on the item you want to change, and then use the Delete (X), up, down, left, or right arrow buttons to reposition the item, if necessary.

Figure 8-5 shows an example. I've just created the last child element, and changed its `Text` property to show "About us." You can't see the entire `NavigateUrl`. It's the path to the `AboutUsHome.aspx` page from the site's root folder, as given here:

```
~/PublicPages/AboutUs/AboutUsHome.aspx
```



You can see both `Default.aspx` and `AboutUsHome.aspx` in Figure 8-1.

Remember, the same technique works whether you're creating a `Menu` control or `TreeView` control. If you ever need to change the control, just open the page on which you placed the control. From the control's Common Tasks menu, choose Edit MenuItems again. In the dialog box, click the + sign next to the root item to expand the menu.



Both the `Menu` and `TreeView` controls have an Auto Format option on their Common Tasks menu. Be sure to check out each one.

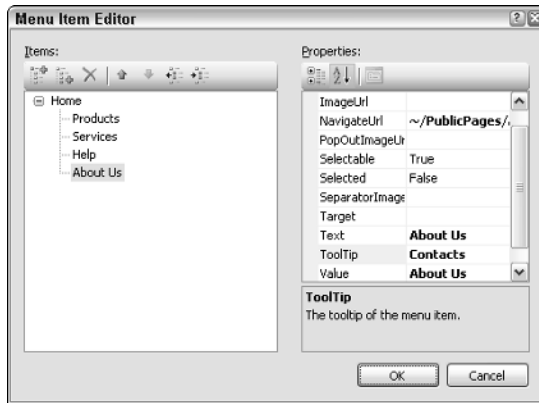


Figure 8-5:
All items in
the Menu
Editor.

As mentioned, using static data with a `Menu` or `TreeView` control is just one way to create a site-navigation menu. The other is to store all the site-navigation info in site map file, and then bind the control to the site map.

Creating a Site Map

There are several ways to create site maps. The easiest is to just create a `Web.sitemap` file in the root of your folder. Then edit the resulting XML file to define your site's navigation structure. Here are the steps to get started:

- 1. Right-click your site name at the top of Solution Explorer and choose Add New Item.**
- 2. In the Add New Item dialog box, click Site Map.**
- 3. Click the Add button.**

In Solution Explorer, you see the `Web.sitemap` name at or near the bottom of the folder hierarchy. On the Design surface, you see an almost-empty site map file with some placeholders for typing text, like in Figure 8-6. (There is no Design view for the `Web.sitemap` file.)

The trickiest thing about using the site map file is keeping track of parents and children. It's all done with nesting (indentations have nothing to do with it). Basically, any element that has child elements must enclose its child elements in a pair of `<siteMapNode>...</siteMapNode>` tags. An element that has no children can be expressed in a single `<siteMapNode />` tag (note the `/>` rather than just `>` at the end of the tag).

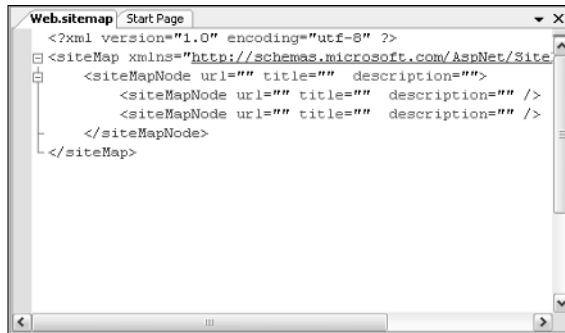


Figure 8-6:
A new Web.sitemap file.

The organization of the XML tags defines the organization of options displayed in a Menu or TreeView control. In XML, it's all about *nesting*. Children of elements must be nested within their parent element. To illustrate, let's start with a simple example.

At the top of Figure 8-7 you see a Web.sitemap file. The outermost <siteMapNode>...</siteMapNode> tags define the root element — that is, the item that appears above all others — of the menu. That root element is the parent to several child elements within the tags. Each of those represents a single link on a “submenu” below the parent. (I removed the url= and description= elements that normally appear in the siteMapNode tags for clarity.)

At the bottom of Figure 8-7 I show how that SiteMap file looks in a Menu control (left) and a TreeView control (right). In both cases, the Home link is the root element at the top of the heap. Each child element is an option “beneath” the root element (and to the right in the case of the Menu control).

In real life, you couldn't just omit the url= attribute, as that's what binds the link to a page in your site. The description= attribute just defines the ToolTip that appears when the user points to the menu item. Here's a sample Web.sitemap with all the attributes in place:

```
<sitemap... >

  <siteMapNode url="~/Default.aspx" title="Home" description="Go home">

    <siteMapNode url="~/PublicPages/Services/ServicesHome.aspx"
      title="Products" description="Product catalog" />

    <siteMapNode url="~/PublicPages/Products/ProductsHome.aspx"
      title="Services" description="Service options" />

  </siteMapNode>
</sitemap>
```

```

    <siteMapNode url="~/PublicPages/Help/HelpHome.aspx"
                title="Help" description="Site help" />

    <siteMapNode url="~/PublicPages/AboutUs/AboutUsHome.aspx"
                title="About Us" description="Contact us" />

</siteMapNode>

</siteMap>

```

Web.sitemap file

```

Parent
├── <siteMap xmlns=... >
│   ├── <siteMapNode title="Home">
│   │   ├── <siteMapNode title="Products" />
│   │   ├── <siteMapNode title="Services" />
│   │   ├── <siteMapNode title="Help" />
│   │   └── <siteMapNode title="About Us" />
│   └── </siteMapNode>
└── </siteMap>
Children

```

Figure 8-7:
A Web.
sitemap file
(top) in
Menu and
TreeView
controls.

```

Home ▶ Products      Home
      Services
      Help
      About Us
Menu control

Home
  Products
  Services
  Help
  About Us
Treeview control

```

Bound controls

As you might imagine, the more complicated the navigational structure of your site, the more complicated the `Web.sitemap` file. For example, Figure 8-8 shows the `Web.sitemap` file (again using only `title=` attributes for brevity) I used to make the `TreeView` control back in Figure 8-2. The braces won't be visible on your screen. I added those to highlight the nesting levels.

The nesting levels in the `Web.sitemap` file define the nesting levels (or show/hide levels) in the `Menu` or `TreeView` control to which you bind the site map.

Typing site maps isn't exactly fun. There's not much in the way of IntelliSense support or other user-friendly features. Writing a `Web.sitemap` file probably involves as much copying and pasting as it does typing.

Figure 8-8:
Web.site
map file
with three
nesting
levels.

```

<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode title="Living Things">
    <siteMapNode title="Animals">
      <siteMapNode title="Mammals">
        <siteMapNode title="Cats" />
        <siteMapNode title="Dogs" />
        <siteMapNode title="Llamas" />
      </siteMapNode>
      <siteMapNode title="Reptiles">
        <siteMapNode title="Turtles" />
        <siteMapNode title="Lizards" />
      </siteMapNode>
    </siteMapNode>
    <siteMapNode title="Plants">
      <siteMapNode title="Fruits">
        <siteMapNode title="Apples" />
        <siteMapNode title="Bananas" />
      </siteMapNode>
      <siteMapNode title="Vegetables">
        <siteMapNode title="Peas" />
        <siteMapNode title="Carrots" />
        <siteMapNode title="Corn" />
      </siteMapNode>
    </siteMapNode>
  </siteMapNode>
</siteMap>

```

The + and – signs at the left side of the editing window expand and collapse nodes within the site map. With a little practice, you can use those buttons to help make sure things are nested properly in your file.

The upside to creating a Web.sitemap file is that the site’s navigation structure is defined in one file. You don’t have to worry about broken hyperlinks all over your site whenever you make a change. When you change your site, you need only change the navigation structure in the Web.sitemap file to match the new structure.

Customizing navigation for roles

If your site has members and roles, you may want different navigation maps for different types of users. For example, the navigation options for an anonymous user might include links to public information only. The options on the navigation menu for an authenticated user, or user in a specific role, might include links to members-only content.

Hiding navigational options from anonymous users is called *security trimming*, because you “trim things out” of a menu by making some options available only to people in specific roles.

Using security trimming in your site menus is a two-step process. First, you have to enable security trimming for your site as a whole. That involves specifically defining `Web.sitemap` as your site's default site map file, and then enabling security trimming on that file by manually editing your site's `Web.config` file. Here are the necessary steps:



1. Open (double-click) the `Web.config` file in your site's root folder.

Don't confuse `Web.config` with `Web.sitemap`. They're two separate files that play two separate roles.

2. Scroll down to the bottom of the `Web.config` file, and get the cursor to a blank line just above the closing `</system.web>` tag.

3. Type the following tags exactly as shown:

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

Type carefully because typing something that's sorta like the above won't cut it. Use the IntelliSense menus as much as possible to minimize typos.

4. Close and save the `Web.config` file.

All you've done so far is change your site's overall configuration a bit to support security trimming in the `Web.sitemap` file. To take advantage of that new feature, you have to specify who can see what in the `Web.sitemap` file. You do so by adding the following attribute to any tag that's to be viewed by members of a specific role only:

```
roles="roleName"
```

The `roleName` must be the name of a role you've previously defined for your site using the Web Site Administration Tool. For example, in Chapter 3, I created a role named `SiteMembers`. So, in my `Web.sitemap` folder, I'd add

```
roles="SiteMembers"
```

to the `sitemapnode` tag of any menu item that should be visible only to logged-in `SiteMembers`. The following is an example of a `Web.sitemap` file, where three of the menu items are visible only to people in the `SiteMembers` role. I've boldfaced the title attributes and new `roles="SiteMembers"` attributes for clarity:

```

<?xml version...>
<siteMap xmlns...>

  <siteMapNode url="-/Default.aspx" title="Home" description="Go home" >

    <siteMapNode url="-/PublicPages/Services/ServicesHome.aspx"
      title="Products" description="Product catalog" />
    <siteMapNode url="-/PublicPages/Products/ProductsHome.aspx"
      title="Services" description="Service options" />

    <siteMapNode url="-/MemberPages/MemberServices/MemberServices.aspx"
      title="My Page" description="Member Services"
      roles="SiteMembers"/>
    <siteMapNode url="-/MemberPages/Forums/ForumsHome.aspx"
      title="Forums" description="Discussions"
      roles="SiteMembers"/>
    <siteMapNode url="-/MemberPages/Downloads/DownloadsHome.aspx"
      title="Download" description="Download stuff"
      roles="SiteMembers" />

    <siteMapNode url="-/PublicPages/Help/HelpHome.aspx"
      title="Help" description="Site help" />
    <siteMapNode url="-/PublicPages/AboutUs/AboutUsHome.aspx"
      title="About Us" description="Contact us" />

  </siteMapNode>
</siteMap>

```

When that `Web.sitemap` file is bound to a navigation `Menu` control, anonymous users see the drop-down menu at the left in Figure 8-9. Authenticated users in the `SiteMembers` role see the navigation menu shown at the right in that same figure.

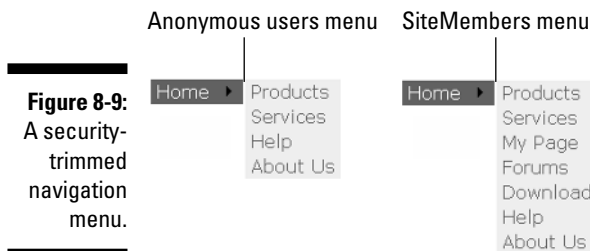


Figure 8-9:
A security-trimmed navigation menu.

Binding a control to `Web.sitemap`

After you've created a `Web.sitemap` file in your site's root folder, you can *bind* it to either a `Menu` or `TreeView` control. "Binding" just means that the control gets its information from the `Web.sitemap` file rather than properties defined

within the control. First, in Design view, you open the Master Page (or .aspx page) on which you want to put the control. Then follow these steps:

1. Drag a **Menu** or **TreeView** control onto the page.
2. From the **Data Source** drop-down list on the **Common Tasks** menu, choose **<New Data Source...>**.
3. In the **Data Source Configuration Wizard** that opens, click **Site Map**.
4. Click **OK**.

That's it. To see the results, close and save the page. Then view the page in a Web browser. (Or, if you put the control on a Master Page, open any page that uses that master.) A **Menu** control initially shows only its root link. When you point to that item, the submenu items appear. To go to any page, click its link.



If there's a problem with your **Menu** or **TreeView** control, the most likely culprit is the `Web.sitemap` file: possibly something as minor as a missing quotation mark or improperly nested tags. If you enabled security trimming, the problem could be in the `Web.config` file.

Adding an Eyebrow Menu

A *breadcrumb* or *eyebrow menu* is a short navigational path back to the home page, usually shown at the top of a page. You see examples of them at many large Web sites, including the Microsoft Developer Network Web site (www.msdn.com). Such a menu usually looks something like this:

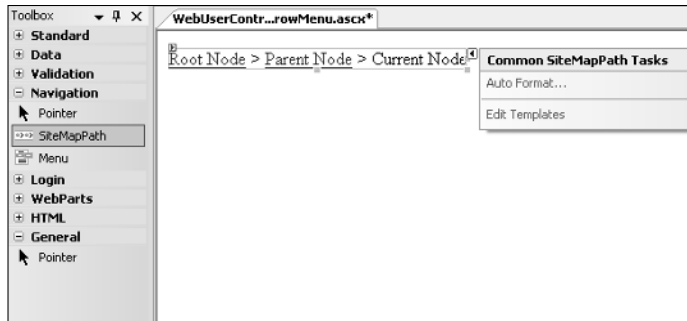
```
MSDN Home > ASP.NET Home > Get Visual Web Developer
```

The path provides a quick view of where the user is in the navigational hierarchy, as well as quick links up the navigational hierarchy.

If your site has a `Web.sitemap` file, you can easily add a breadcrumb menu to the top of any page in your site. Just drag a `SiteMapPath` tool from the Toolbox onto your page. It appears as a generic eyebrow menu in Design view as in the example shown in Figure 8-10.

Like all server controls, `SiteMapPath` has a **Common Tasks** menu with an easy **AutoFormat** option. It also has an extensive **Properties** sheet where you can change things like font, color, and so forth. But you don't have to do anything to it if you don't want. Just close and save the page.

Figure 8-10:
Use a **SiteMap Path** control for eyebrow menus.



Creating Web User Controls

Though not directly relevant to site navigation, I'd be remiss in my duties if I didn't make you aware of Web User Controls. Here's why: In any given Web site there's likely to be stuff you want to show on every page. That stuff you can put in a Master Page. On the other hand, there may also be stuff that you want to put on some, but not all, pages. An eyebrow menu would be a good example, because such a menu doesn't really make sense on pages that are at the top of a navigational hierarchy.

If you drag-and-drop a control straight from the Toolbox onto a page, you create a whole new control. If you want consistency across your site, you'll have to style the control exactly the same on every page in your site. If you ever change your mind about that style, you'll have to make the change to every page that uses the control. Bummer.

The simple solution to the problem is a Web User Control. Getting back to the eyebrow menu example, you could put the `SiteMapPath` control in a Web User Control, and design it as you see fit, and then close and save the Web User Control.

Then, any time you want an eyebrow menu on the page you're editing, drag the Web User Control from Solution Explorer onto the page (which I named `MyEyebrowMenu.ascx` in Figure 8-11). Don't use the generic `SiteMapPath` control from the Toolbox anymore, as that one won't have the same style as the one in the Web User Control.

When you drop the Web User Control onto your page, you won't have to design it because it's already been designed. And, if you ever decide to change the appearance of the control that's inside the Web User Control, no problem. Just open the Web User Control in Design view, make your changes, and close and save the page. The change is automatically reflected in every page on which you've placed the Web User Control.

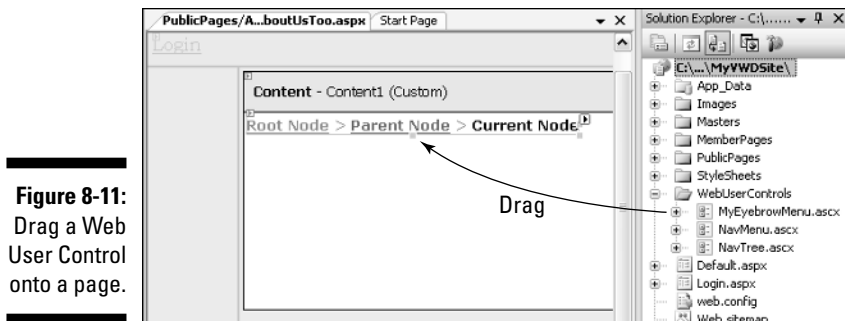


Figure 8-11:
Drag a Web
User Control
onto a page.

And remember, the eyebrow menu is just an example. Web User Controls are perfect for *anything* that you might want to show on some, but not all, pages throughout your site.

A good starting point might be to create a folder for the controls. To do so, just right-click the site name at the top of Solution Explorer and choose Add Folder → Regular Folder. Give the folder a name (I'll name mine WebUserControls) and press Enter.

Creating a Web User Control

Creating a Web User Control is almost identical to creating a regular Web Form page. Here are the steps:

- 1. In Solution Explorer, right-click the folder in which you want to store the control and choose Add New Item.**
- 2. In the Add New Item dialog box, click Web User Control.**
- 3. Enter a filename of your choosing.**
I named mine `MyEyebrowMenu.ascx` in Figure 8-12.
- 4. Choose language options to taste.**
I chose the usual in Figure 8-12.
- 5. Click Add.**

The new control opens looking just like an empty Web page (in Design view). And basically, it is just an empty page in the sense that you can put whatever you want into it. In the eyebrow menu example, you'd drag a `SiteMapPath` control from the Toolbox onto the page. But any server control from any Toolbox category would be fine as well.

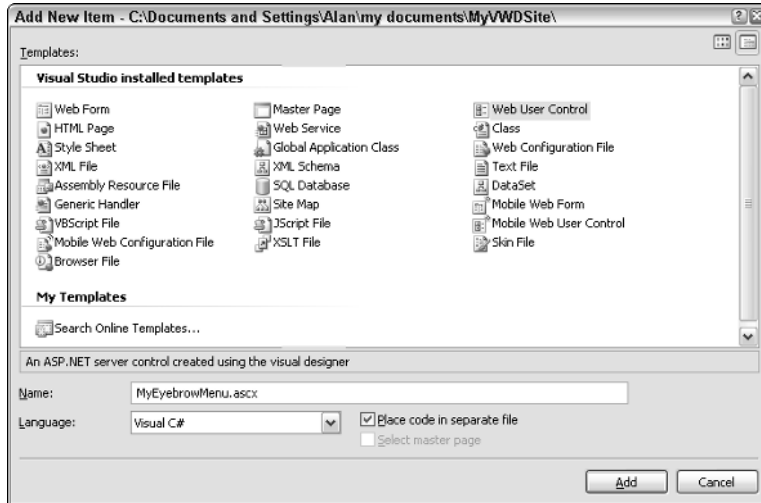


Figure 8-12:
Creating a
Web User
Control.

You can style the item using its Common Tasks and Properties menus. Or you can leave it as-is. Then just close and save the page. Its filename extension, `.ascx`, identifies it as a Web User Control.



Unlike a page, you can't view a Web User Control in a Web browser. You first have to put the Web User Control on a page, and then open that page in a Web browser.

Using a Web User Control

Any time you want to display a Web User Control on one of your pages, just drag its filename from Solution Explorer onto your page, as in the example shown back in Figure 8-11. Nothing else is required. To see the control, close and save the page on which you placed the control. Then view that page in a Web browser.

Should you decide to change the style of the control that's inside the Web User Control, here's the drill:

- 1. Open the Web User Control by double-clicking its name in Solution Explorer.**
- 2. Edit the control in Design view as you normally would.**
- 3. Close and save the control.**



All pages that use the Web User Control display the control with your current format settings.

Don't be shy with Web User Controls

You can also create different Web User Controls for different types of site visitors. For example, you could create a Web User Control named `Anon.ascx` (or whatever), and put whatever you want in that control. Create another Web User Control named `Authenticated.ascx` (or whatever), and put whatever you want in that one. Close and save both controls.

Then, on any page where you want to show one control or the other, first drag a `LoginView`

control onto the page. Then drag `Anon.ascx` into the Anonymous Template of that control. Switch to the Logged In template, and drag the `Authenticated.ascx` file into that template on the `LoginView` control. Anonymous users will see only what's in `Anon.ascx`. Authenticated users will see only what's in `Authenticated.ascx`.

Part III

Personalization and Databases

The 5th Wave

By Rich Tennant



"Your database is beyond repair, but before I tell you our backup recommendation, let me ask you a question. How many index cards do you think will fit on the walls of your computer room?"

In this part . . .

If you want users to return to your site regularly, you have to give them a sense of belonging to a community. One way to accomplish that is to give them the option of personalizing their use of your site to their liking. In this part, you get a handle on using the Personalization and Themes features of VWD to give users what they want.

If you plan to offer courses, products, or other things that users might purchase or sign up for, your site will need some custom database tables to store data about your offerings and record transactions. This part unravels the mysteries about how to create these features.

Chapter 9

Using Personalization

In This Chapter

- ▶ Creating user profiles
 - ▶ Storing information about users as profile properties
 - ▶ Retrieving, changing, and saving profile properties
 - ▶ Using validation controls
 - ▶ Using the Forms Designer
-

When you configure your Web site to support membership, VWD automatically creates database tables to store information about user accounts. However, it creates only the bare-minimum number of fields needed, such as `User Name`, `E-mail Address`, and `Password`. If you want to store more information than that about each user, such as name, address, and phone number, you must define *profile properties*.

The basic idea in VWD is this: Every authenticated user has a profile that contains information about that user. In code, you can use the simple syntax `Profile.propertyname` to get, or store, information about each user. The keyword `Profile` (with an uppercase P) always means “whichever user happens to be viewing this page.”

The first step is to decide what properties you want to store for each user — and then configure your site to support those profiles.

Creating a User Profile

A user profile consists of a set of property names, where each property name represents the unit of information you want to store. The names you give your properties are entirely up to you. Just keep the name short, no blank spaces, and make sure it starts with a letter (A–Z). Some common examples include names like `FirstName`, `LastName`, `Address`, `City`, `State`, and `ZIPCode`.

Why strings for phone numbers and ZIP codes?

It may seem that phone numbers and ZIP codes should be numbers rather than strings. After all, aren't (215) 555-1234 and 00123-1343 numbers? Actually, they're not numbers. At least, they're not *scalar values* that represent some quantity. With scalar values, you can do math (addition, subtraction, multiplication, and division) to come up with some new meaningful result. Try doing math with ZIP codes or phone numbers, and even if you can come up with some sort of result, that result has no real meaning.

Also, numbers must start with a numeric digit (0–9) or a leading minus sign. A number cannot contain parentheses or embedded hyphens. Hence, you couldn't type a phone number like (215) 555-1234 or a ZIP code like 00123-1343 into a numeric field even if you wanted to. You only want to use the `Number` data types for true scalar values such as 10, 98.6, -32, 9.99, and so forth.

You can also choose a *data type* for each field — that is, a description of the type of information being stored, whether it's text (string), a number, a date, whatever.



If you don't know a data type from a turtledove, suffice it to say that the `String` data type can store just about anything. So if in doubt, use the `String` data type.

For the sake of example, in this chapter we create eight profile properties to store the following information about each person who creates an account on your Web site:

- ✓ **FirstName:** This stores the person's first name (such as Joe or Mary).
- ✓ **LastName:** This stores the person's surname (such as Smith or Jones).
- ✓ **Address1:** First line of street address.
- ✓ **Address2:** Optional second line of street address.
- ✓ **City:** Town or city in which the person lives.
- ✓ **StateProvince:** State or province in which the person lives.
- ✓ **ZIPPostalCode:** ZIP code or postal code in which the person lives.
- ✓ **Country:** Country where the person lives.

You can assign *default values* to properties. For example, you can set the default value of the `Country` field to USA if you think most site members will be American. The default value is simply the value that's stored in the property unless the user specifies another.

Setting up user profiles

To define user profiles, you need to manually edit the `Web.config` file in the root of your Web site. Here's how:

1. Open `Web.config` for editing by just double-clicking its icon.

Typically you'll see that filename located near the bottom of Solution Explorer.

2. Just above the closing `</system.Web>` tag, type

```
<profile>
```

and press Enter. A closing `</profile>` tag is added automatically. So you end up with:

```
<profile>
```

```
</profile>
```

3. With the cursor placed between the `<profile>` and `</profile>` tags, type:

```
<properties>
```

and press Enter. Once again, the closing `</properties>` tag is entered automatically. So you end up with

```
<profile>
  <properties>

  </properties>
</profile>
```

4. Between the `<properties>` and `</properties>` tags you need to type a tag for each property you want to define using the following syntax:

```
<add propertyName />
```

where *propertyName* is the name of the profile property.

If you're using the `string` data type, as all the sample fields described earlier were, then you don't need to specify a data type. So in this example all you'd really need in your `Web.config` file is the following:

```
<profile>
  <properties>
    <add name="FirstName" />
    <add name="LastName" />
    <add name="Address1" />
    <add name="Address2" />
    <add name="City" />
    <add name="StateProvince" />
    <add name="ZIPPostalCode" />
    <add name="Country" />
```



```
</properties>  
</profile>
```

When you're typing code, like here, there is no margin for error. You must type every character and blank space correctly, or you'll get error messages when you try to view a Web page in your browser.

If you want to put in a default value for a property, include a `defaultValue="value"` in the tag (where `value` is the text you want inserted into the property automatically). For example, to make USA the default value for the Country property, add `defaultValue="USA"` to the tag that defines the Country property, as shown below:

```
<profile>  
  <properties>  
    <add name="FirstName" />  
    <add name="LastName" />  
    <add name="Address1" />  
    <add name="Address2" />  
    <add name="City" />  
    <add name="StateProvince" />  
    <add name="ZIPPostalCode" />  
    <add name="Country" defaultValue="USA" />  
  </properties>  
</profile>
```

I should mention that even though you *can* omit the data type when defining string value, that doesn't mean you *must* omit them. When you look at other examples of profile properties, you might see the `string` data type defined explicitly, like this:

```
<profile>  
  <properties>  
    <add name="FirstName" type="System.String" />  
    <add name="LastName" type="System.String" />  
    <add name="Address1" type="System.String" />  
    <add name="Address2" type="System.String" />  
    <add name="City" type="System.String" />  
    <add name="StateProvince" type="System.String" />  
    <add name="ZIPPostalCode" type="System.String" />  
    <add name="Country" defaultValue="USA"  
      type="System.String" />  
  </properties>  
</profile>
```

Try not to let this confuse you. Including the `type="System.String"` attribute really has no effect — even if you omit the attribute, the data type will be `System.String` (called a `string` for short).

So, just to make sure we're all on the same page here, let's recap. Assuming you added the same properties described in this chapter, your entire `Web.config` file should look something like the example in Figure 9-1.

Figure 9-1:
Profile
properties in
a sample
Web.
config
file.



```

web.config
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration"
  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms" />

    <profile>
      <properties>
        <add name="FirstName"/>
        <add name="LastName"/>
        <add name="Address1"/>
        <add name="Address2"/>
        <add name="City"/>
        <add name="StateProvince"/>
        <add name="ZIPPostalCode"/>
        <add name="Country" defaultValue="USA"/>
      </properties>
    </profile>
  </system.web>
</configuration>

```

If you happened to add the tags for relaxing password rules as described in Chapter 7, your Web.config file might look more like the one in Figure 9-2.

When you're sure you have it right, just close and save the Web.config file.

Figure 9-2:
Another
sample
Web.
config
file
with profile
properties
defined.



```

web.config
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <roleManager enabled="true" />
    <authentication mode="Forms" />
    <membership>
      <providers>
        <remove name="AspNetSqlMembershipProvider"/>
        <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider"
          connectionStringName="LocalSqlServer"
          minRequiredPasswordLength="5"
          minRequiredNonalphanumericCharacters="0"
          passwordStrengthRegularExpression="" />
      </providers>
    </membership>
    <profile>
      <properties>
        <add name="FirstName"/>
        <add name="LastName"/>
        <add name="Address1"/>
        <add name="Address2"/>
        <add name="City"/>
        <add name="StateProvince"/>
        <add name="ZIPPostalCode"/>
        <add name="Country" defaultValue="US&"/>
      </properties>
    </profile>
  </system.web>
</configuration>

```

Letting Users Enter Properties

The whole purpose of creating profile properties is to store information about users (site members). You wouldn't want to type in all that information about each user yourself. Rather, you want each user to type in his or her own profile properties. To do so, you need to provide users with a fill-in-the-blanks *form*.

A form is basically a Web page with controls for entering and editing data (information). To create a Web form for entering profile properties, just create a new, empty .aspx page. In the following steps, I create a page named `CreateProfile.aspx` to let users enter profile information in their own accounts:

1. In Solution Explorer, right-click the folder in which you want to store the new page and choose **Add New Item**.

In my example, I right-clicked my `MemberPages` folder.

2. In the **Add New Item** dialog box, click **Web Form**.

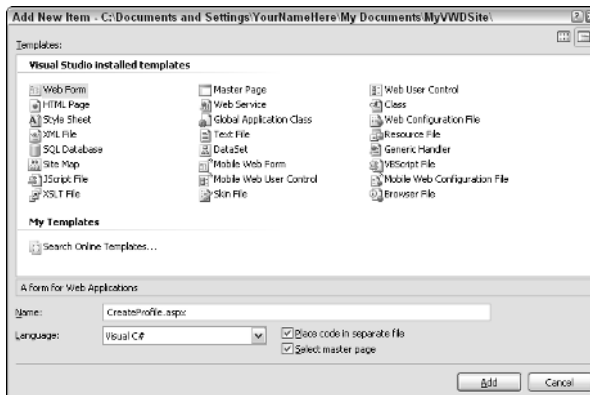
3. Enter a filename for your page.

I entered `CreateProfile.aspx` for my page.

4. Choose other options to taste.

I chose the options shown in Figure 9-3.

Figure 9-3:
About to create a Web Form named `CreateProfile.aspx`.



5. Click the **Add** button.

6. If you opted to use a **Master Page**, choose your master, and then click **OK**.

A new, blank page (or content placeholder) opens ready for you to design your form. If it opens in Source view, click the Design button at the bottom of the Design surface to switch to Design view. Then you can add text, pictures, links, and controls to the page using all the standard methods described in earlier chapters.

Here, some `Textbox` controls will allow users to enter profile information on this new page. Probably your best bet would be to add a table to the page first, to make it easy to organize the text boxes. The left column of the table would be used for plain-English descriptions of what you expect the user to type in the text boxes in the right column, as shown in the example in Figure 9-4.

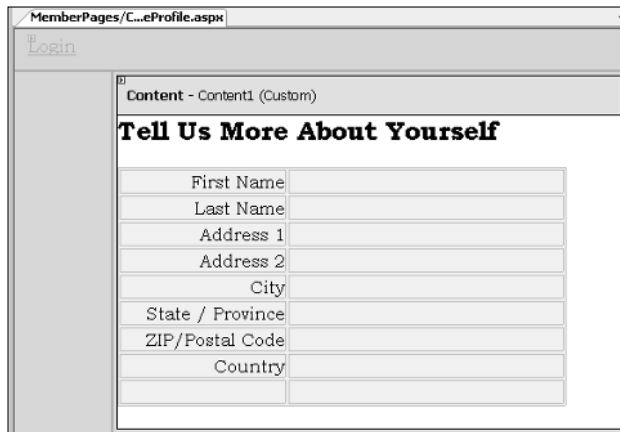
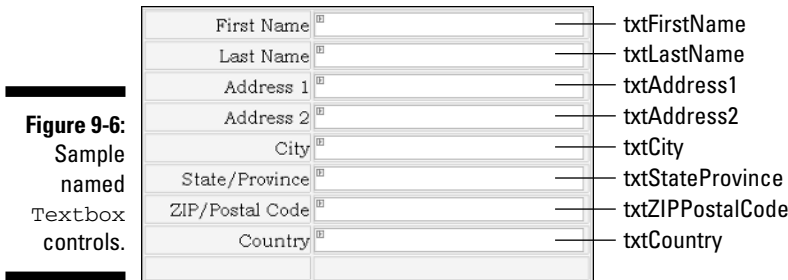
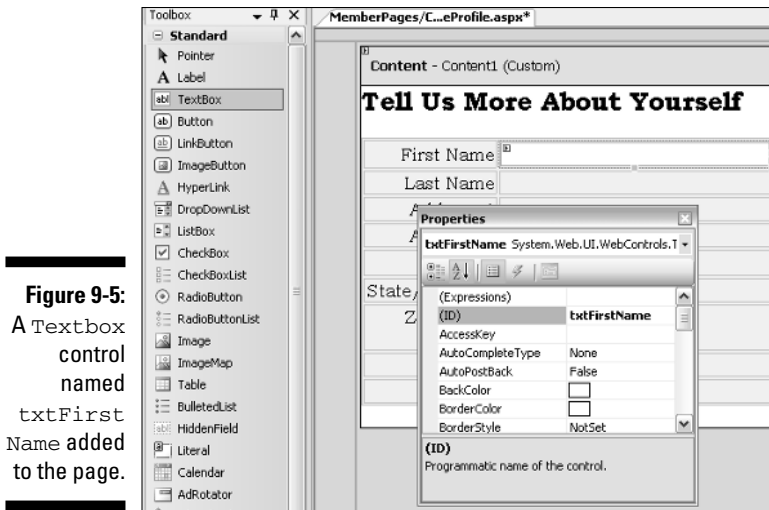


Figure 9-4:
Table and
some text
added to a
page.

To make the table act as a form, you'll need to add some `Textbox` controls so users can type in their own information. You'll also need to assign a *programmatic name* to each control, via the control's `ID` property. You'll use the programmatic name to refer to the text box from code. The programmatic name can be just about anything you want, but must start with a letter and cannot contain spaces or punctuation marks.

For example, in Figure 9-5 I dragged a `Textbox` control from the Standard tools (visible in the Toolbox at the left side of the figure) into a table cell. The text box is the white rectangle. That control is currently selected so you can see its Properties sheet. As you can see in the figure, I've named the text box `txtFirstName` by typing that name as the control's `ID` property in its Properties sheet.

You'll need to add (and name) a `Textbox` control for each profile property that you want users to see or edit. For my example, I created eight `Textbox` controls and named them as shown in Figure 9-6. Adding "txt" to the front of each name isn't a technical requirement. I just stuck that on there to remind myself later that the name refers to a `Textbox` control.

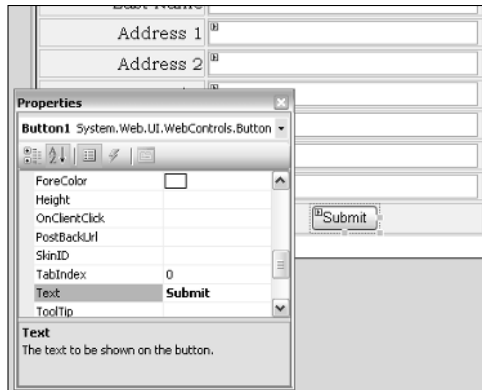


Adding a button

In addition to having some blanks to fill in, users need a button to indicate when they're done typing and ready to submit their information. To add a button to the form, just drag a `Button` control from the Standard tools onto the form. VWD automatically names the button `Button1`. You can keep that name or change it by changing the button's `ID` property. For my example, I'll keep the default name.

To change the text that appears on the button, change the button's `Text` property. Figure 9-7 shows an example where I've added a button to my sample form, and set its `Text` property to `Submit`. So the button shows `Submit` as its label. (You could use `OK` or anything else you want for the button's `Text` property.)

Figure 9-7:
A button
added to my
sample
form.



So now you have a page that shows some empty text boxes, and a button that does nothing when you click it. But it has no connection whatsoever to profile properties. Before this page is of any real value, I need to come up with a way to copy whatever the user types into the `Textbox` controls into that user's profile properties. That requires some programming. And programming means writing code in the page's code-behind file.

Writing some code

Programming is all about writing *code* — which is slang for “computer instructions written in a programming language like C# or Visual Basic” (and not for “hieroglyphics,” no matter what you’ve heard). Learning to write code fluently is an enormous undertaking and usually requires several months, if not years, of study and experience to master. I can’t get into all of that here; there’s simply not enough room in this book. But I can tell you exactly what’s needed in terms of using profile properties.

In code, if you want to refer to the contents of a `Textbox`, you use the syntax

```
ctrlName.Text
```

where `ctrlName` is the programmatic name (ID) of the control. For example, in code, `txtFirstName.Text` refers to the contents of the `txtFirstName` text box.

To refer to a specific profile for the current user, the syntax is:

```
Profile.propertyName
```


where *propertyName* is one of the property names you created yourself. For example, `Profile.FirstName` refers to the first name of whatever user happens to be viewing the page at the moment.

To copy the contents of a text box into a profile property, make the profile property “equal to” the `Textbox` control’s contents. The C# syntax for that looks like this:

```
Profile.propertyName = txtBoxID.Text;
```

Here *propertyName* is one of your profile property names and *txtBoxID* is the ID of its corresponding text box. The semicolon (;) at the end of the line is a C# requirement. In Visual Basic, you just omit the semicolon.



If you forget the semicolon, the C# Editor shows a little red squiggle where the semicolon should be. Type in the semicolon and your code should work properly.

The following is a line of C# code that copies the contents of the `txtFirstName` control to the current user’s `Profile.FirstName` property:

```
Profile.FirstName = txtFirstName.Text;
```

To copy the contents of every text box to the user’s `Profile` properties, you need several lines of code, like this:

```
Profile.FirstName = txtFirstName.Text;  
Profile.LastName = txtLastName.Text;  
Profile.Address1 = txtAddress1.Text;  
Profile.Address2 = txtAddress2.Text;  
Profile.City = txtCity.Text;  
Profile.StateProvince = txtStateProvince.Text;  
Profile.ZIPPostalCode = txtZipPostalCode.Text;  
Profile.Country = txtCountry.Text;
```

Tying code to an event

To ensure that the code is executed only when the user clicks the Submit button, tie the code to the `Click` event for `Button1`. The easy way to tie code to a button is to simply double-click the button in Design view. The code-behind page (where all code is kept) for the page opens, looking something like Figure 9-8.

The code-behind page is the place where you put the logic of a page — the code that tells computers what to do, rather than stuff the user actually sees on the screen. After it’s opened, the code-behind page appears in its own tabbed document window. You can switch back and forth between the `.aspx` page and the code-behind page by clicking their tabs.

Figure 9-8:
Code-
behind page
for my
Create
Profile
example.

```

1  using System;
2  using System.Data;
3  using System.Configuration;
4  using System.Collections;
5  using System.Web;
6  using System.Web.Security;
7  using System.Web.UI;
8  using System.Web.UI.WebControls;
9  using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11
12 public partial class CreateProfile : System.Web.UI.Page
13 {
14     protected void Page_Load(object sender, EventArgs e)
15     {
16     }
17
18     protected void Button1_Click(object sender, EventArgs e)
19     {
20     }
21 }
22
23

```



When using C# as your programming language, the code-behind page has the same name as the .aspx page followed by a .cs extension. For example, the code-behind page for `CreateProfile.aspx` is named `CreateProfile.aspx.cs`.

At the top of the code-behind page is a bunch of `using` directives followed by more C# code. Basically, you don't want to change or remove anything that's in the page unless you *really* know what you're doing. Just now, all you want to do is add some code to the `Button1_Click` event handler — which appears near the bottom of the page and looks like this:

```
protected void Button1_Click(object sender, EventArgs e)
{
}

```



Do not confuse the `Button1_Click` procedure with the `Page_Load` procedure that precedes it. You'll use the `Page_Load` procedure later. Make sure you get the cursor between the correct procedure's opening and closing curly braces before you start typing.

Type the properties code you created in the previous section here. When typing code, there is no margin for typographical errors. So you want to use IntelliSense as much as possible to minimize those errors. For example, when typing the first line, you can type `Prof` and then press `Enter` to choose `Profile` from the IntelliSense menu.

Next, type the dot (period), and the IntelliSense menu shows valid words/names you can type there, as in Figure 9-9. Again, to get the word you want, you can just keep typing, press Enter when the highlighter is on the word you want in the IntelliSense menu, or double-click the word you want in the IntelliSense menu.



Yellow highlights near line numbers indicate lines that have been added or changed but not yet saved.

```

12 public partial class CreateProfile : System.Web.UI.Page
13 {
14     protected void Page_Load(object sender, EventArgs e)
15     {
16     }
17 }
18     protected void Button1_Click(object sender, EventArgs e)
19     {
20         Profile.F
21     }
22 }
23
  
```

Figure 9-9:
Sample
IntelliSense
menu.



When typing C# code, remember to type a semicolon (;) at the end of each statement. If you forget the semicolon, the editor gives you a little red squiggly line showing where to put it.

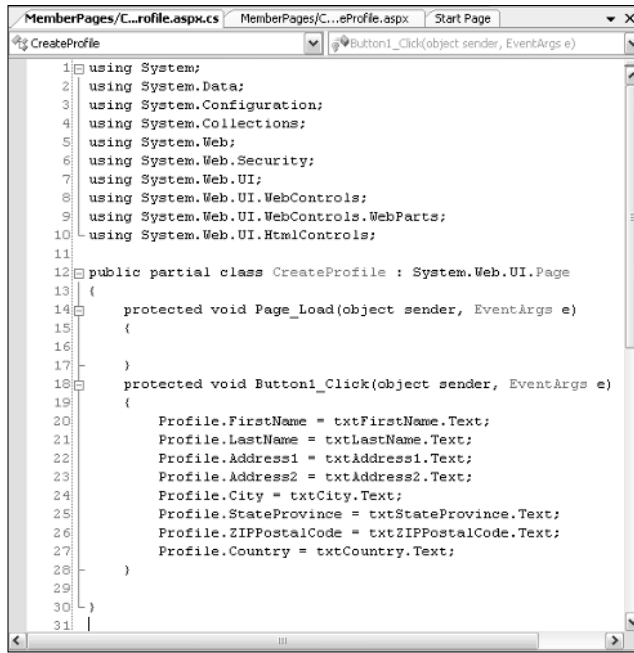
When you've finished typing all the code, your code-behind page should look something like Figure 9-10.

The wrong page opened!

Normally you can view any page in your site by right-clicking its name in Solution Explorer and choosing View in Browser. However, when you start using privileged content in your site, it doesn't always work that way. By default, when an anonymous user attempts to access a page that requires authentication, they're automatically redirected to the login page.

When you right-click a protected page in Solution Explorer and choose View in Browser, the same thing happens if you're not logged in — you're instantly redirected to your own login page, making it look as though the wrong page has opened! To get around this problem, you simply log in to a valid user account and then open the restricted page.

Figure 9-10:
Here's the
Button1_
Click
event
handler,
ready to go.



```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11
12 public partial class CreateProfile : System.Web.UI.Page
13 {
14     protected void Page_Load(object sender, EventArgs e)
15     {
16     }
17
18     protected void Button1_Click(object sender, EventArgs e)
19     {
20         Profile.FirstName = txtFirstName.Text;
21         Profile.LastName = txtLastName.Text;
22         Profile.Address1 = txtAddress1.Text;
23         Profile.Address2 = txtAddress2.Text;
24         Profile.City = txtCity.Text;
25         Profile.StateProvince = txtStateProvince.Text;
26         Profile.ZIPPostalCode = txtZIPPostalCode.Text;
27         Profile.Country = txtCountry.Text;
28     }
29 }
30
31
```

When you're finished typing your code, just close and save the file as you would any other, by clicking the Close (X) button in its upper-right corner. The original .aspx page will still be open. The code you wrote doesn't affect its appearance or behavior in Design view. The code won't actually do anything until you open the page in a browser.

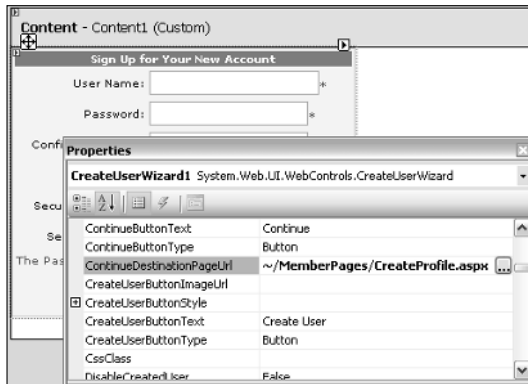
Determining where to put the profile information

The next thing to consider is *when* users enter profile information. Of course, you can provide a link to the page from any place you want (though a good time to grab users is right after they finish successfully creating their individual user accounts). If you used a `CreateUserWizard` control to let users set up accounts, the job is easy. You just set the `ContinueDestinationPageURL` property of the `CreateUserWizard` control to the `CreateProfile.aspx` page.

For example, back in Chapter 7 I created a page in my sample site named `CreateAcct.aspx`. I would just open that page in Design view, click the control

to select it, and then scroll through the (lengthy) set of properties until I got to `ContinueDestinationPageURL`. There I'd click the property, click the Build button, navigate to my `CreateProfile.aspx` page, and click OK. The property then shows the path to that page, as shown in Figure 9-11.

Figure 9-11:
The
Continue
Destina
tionPage
URL prop
erty of a
Create
User
Wizard
control.



Close and save the `CreateAcct.aspx` page — and that takes care of allowing users to create accounts and enter profile information.

Though I didn't specifically mention it, I was thinking about Web site security the whole way through the above scenario. I don't want anonymous users to have any access to profiles, so I put the `CreateProfile.aspx` page in my `MemberPages` folder. That way an anonymous user can't get to that page. The only way to get to that page is by first successfully creating a valid user account.

The word "successfully" is key there. I wouldn't want people to sneak around setting up accounts. But that's not going to happen; only users who successfully create an account can reach the `ContinueDestinationPageURL`. That restriction is built right into the logic of the `CreateUserWizard` (so I don't have to worry about it).

If you're going to allow users to enter profile information, you'll also have to let them change their own information. I haven't done anything about that yet. But I can do so pretty quickly.

Letting users edit their profiles

So far I've only created a form that can *store* information in a user's profile. We haven't come up with a means to *retrieve* a user's profile properties. If we want users to edit their own profiles, we need a page that can retrieve the profile information, show it on a form for the user to view (or edit), and if need be, send any changes back to the database.

Getting data from the profile to the text box is just the opposite of getting data from the text box to the profile. That is, rather than making the profile property equal to the text box's text, you make the text box text equal to the profile property. For example, to copy the user's `FirstName` profile property to a control named `txtFirstName`, the C# statement is:

```
txtFirstName.Text = Profile.FirstName;
```

You could whip this profile-retrieval page together quickly just by making a copy of `CreateProfile.aspx` and renaming it to `EditProfile.aspx`. To do this, open the `EditProfile.aspx` page, then double-click some empty space on the page background (or the content placeholder). Or you can just double-click `EditProfile.aspx.cs` in Solution Explorer. Either way, the code-behind page opens.

You want the profile properties to be loaded into the page as soon as the page opens so the user sees their information and can make changes. To execute code as soon as a page opens, put that code in the page's `Page_Load` event handler.

There's just one catch. With ASP.NET programming, every time a user does something on your page, a *postback* gets sent to the Web server. The postback causes the `Page_Load` event handler to execute again — which is fine in some cases — sometimes (as in this situation) the `Page_Load` code should execute only when the user first opens the page — not every time they click a button on the page.

To prevent code from being executed on postbacks, you can add some “If not postback, then . . .” C# logic to your `Page_Load` procedure, as follows:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {

    }
}
```



In C#, an exclamation point (!) means *not* — so `if (!Page.IsPostBack){}` is the C# way of saying, “If the page has already been loaded into the user's browser, don't execute the code in my curly braces.” The code you add to copy profile properties to text boxes must go in the `if (!Page.IsPostBack){}` block's curly braces, as shown below:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        txtFirstName.Text = Profile.FirstName;
    }
}
```

```

        txtLastName.Text = Profile.LastName;
        txtAddress1.Text = Profile.Address1;
        txtAddress2.Text = Profile.Address2;
        txtCity.Text = Profile.City;
        txtStateProvince.Text = Profile.StateProvince;
        txtZIPPostalCode.Text = Profile.ZIPPostalCode;
        txtCountry.Text = Profile.Country;
    }
}

```

Figure 9-12 shows how the `EditProfile.aspx.cs` page looks with all the right code in place. (Please note that I clicked the – sign next to the first `using` statement to collapse and hide it. But I did not delete anything up there.)

Now the `EditProfile.aspx.cs` page has two C# procedures, each of which plays a different role in the page:

- ✓ **Page_Load:** Copies the contents of the user’s profile from the Web server database to `Textbox` controls on the user’s Web page. This happens when the user first opens the page.
- ✓ **Button1_Click:** Copies the contents of the `Textbox` controls on the Web page to the user’s profile on the database. This happens when the user clicks the Submit button. If the user never clicks the Submit button, nothing gets copied to the database.

```

MemberPages/E...file.aspx.cs* MemberPages/EditProfile.aspx Start Page
CreateProfile Page_Load(object sender, EventArgs e)
1 using ...
11
12 public partial class CreateProfile : System.Web.UI.Page
13 {
14     protected void Page_Load(object sender, EventArgs e)
15     {
16         if (!Page.IsPostBack)
17         {
18             txtFirstName.Text = Profile.FirstName;
19             txtLastName.Text = Profile.LastName;
20             txtAddress1.Text = Profile.Address1;
21             txtAddress2.Text = Profile.Address2;
22             txtCity.Text = Profile.City;
23             txtStateProvince.Text = Profile.StateProvince;
24             txtZIPPostalCode.Text = Profile.ZIPPostalCode;
25             txtCountry.Text = Profile.Country;
26         }
27     }
28     protected void Button1_Click(object sender, EventArgs e)
29     {
30         Profile.FirstName = txtFirstName.Text;
31         Profile.LastName = txtLastName.Text;
32         Profile.Address1 = txtAddress1.Text;
33         Profile.Address2 = txtAddress2.Text;
34         Profile.City = txtCity.Text;
35         Profile.StateProvince = txtStateProvince.Text;
36         Profile.ZIPPostalCode = txtZIPPostalCode.Text;
37         Profile.Country = txtCountry.Text;
38     }
39 }

```

Figure 9-12:
The
EditProf
ile.aspx
.cs page.

Where are profile properties stored?

The profile properties are stored in the same database as the rest of the membership stuff—the database you see when you click the Database Explorer tab instead of Solution Explorer. Specifically, the profile properties are

stored in the `aspnet_Profile` table. But the profiles aren't stored in a traditional manner, so you don't want to go rummaging around in that table unless you *really* know what you're doing.

That takes care of allowing users to view and edit their own profile information. To put it all to the test, you can start by opening (in a Web browser) the page for creating a user account. Then actually create a new account, click Continue, and you should then be able to add some profile properties to the new account via your `CreateProfile` page. You should also be able to view and edit a user's profile using your `EditProfile` page.



If you put your `CreateProfile` and `EditProfile` pages in a protected folder, don't be surprised if your login page opens when you try to view either one in a Web browser. When you're in a Web browser, you're just another user. Like everyone else, you have to log in to a valid user account before you can access any page in your site's protected folder(s).

Of course, if you made even the tiniest mistake when writing your code, or doing any other step in the overall process, you'll get error messages rather than working Web pages. In that case, you'll need to fix whatever is wrong before you can get the pages to work.

Using profile properties with Visual Basic

I used C# as the language in the above example. But in Visual Basic, the techniques and code are basically the same: Double-click the page background to get to the `Page_Load` event for the page and double-click a button to get to its event handler.

In Visual Basic, you'd use the `If . . . End If` block shown below in place of the C# `if {}` block. The rest of the lines are identical to the C# code, minus the semicolon at the end of each line (prompting some C programmers to refer to C# as “Visual Basic with semicolons”).

```
Partial Class ProfileVB
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
```



```
        ByVal e As System.EventArgs) Handles Me.Load
    If Not Page.IsPostBack Then
        txtFirstName.Text = Profile.FirstName
        txtLastName.Text = Profile.LastName
        txtAddress1.Text = Profile.Address1
        txtAddress2.Text = Profile.Address2
        txtCity.Text = Profile.City
        txtStateProvince.Text = Profile.StateProvince
        txtZIPPostalCode.Text = Profile.ZIPPostalCode
    End If
End Sub

Protected Sub Button1_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
    Profile.FirstName = txtFirstName.Text
    Profile.LastName = txtLastName.Text
    Profile.Address1 = txtAddress1.Text
    Profile.Address2 = txtAddress2.Text
    Profile.City = txtCity.Text
    Profile.StateProvince = txtStateProvince.Text
    Profile.ZIPPostalCode = txtZIPPostalCode.Text
    Profile.Country = txtCountry.Text
End Sub

End Class
```

Using Validation Controls

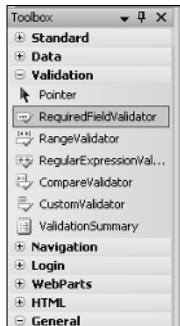
The sample forms I presented earlier in this chapter have one weakness: They accept anything the user types into the boxes. In fact, they accept the form even if the user leaves every text box empty. That might be okay if you don't want to force users to enter personal information. But if getting profile data from users is important, you may have to reject empty fields or fields that contain meaningless data.

The ASP.NET validation controls, available in the Validation section of the Toolbox, make it easy to verify form data (see Figure 9-13). To use a validation control, you first need a place to put it on your form. You'll want to put it near the control you're validating, because the control will display an error message if the user's entry isn't valid.

Most validation controls have certain properties that must be set in order for the validation to work. The main properties are these:

- ✓ **ControlToValidate:** Specify the name (ID) of the `TextBox` control that should be validated.
- ✓ **Display:** Choose `Dynamic` so the error message takes up no space on the screen unless a user's entry fails the validity test.

Figure 9-13:
Validation
controls
hanging
around in
the Toolbox.



- ✓ **ErrorMessage:** Any text you type here is used in a `ValidationSummary` control (if any) on the current page. This is not the error message that appears near the control.
- ✓ **Text:** Whatever you type here appears next to the control, but only if a user's entry fails to pass validation. It can be a simple asterisk (*) or a more descriptive message such as `Required Field!`.

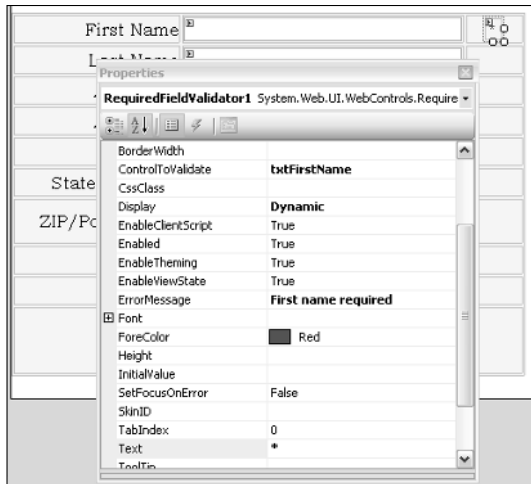
The different kinds of validation controls available to you are described, one by one, in the following sections.

RequiredFieldValidator

The most commonly used validation control is the `RequiredFieldValidator`, which prevents users from leaving a text box empty. It's super-easy to use: You just drag the control to your page, and then set its `ControlToValidate` property to the programmatic name (ID) of the control that you don't want left blank. Figure 9-14 shows an example; here's how I set it up:

- ✓ I added a third column to a table, and then dragged a `RequiredFieldValidator` control to the cell to the right of my `txtFirstName` `Textbox` control.
- ✓ I set the `ControlToValidate` property to `txtFirstName` to ensure that the text box would not be left blank. I set the `Display` property to `Dynamic`, and the `ErrorMessage` to `First name required`.
- ✓ I set the `Text` property to an asterisk (*) — which reduced the size of the control to an asterisk. You can see the control selected, in the upper-right table cell, in Figure 9-14.

Figure 9-14:
Using a
validation
control to
keep the
First Name
control from
being left
blank.



In a Web browser, those properties play out as follows:

- ✓ When the page first opens, the validation control is there but invisible.
- ✓ If the user clicks the Submit button and the text box is empty, the validation control shows its `Text` property (here it's a red asterisk).
- ✓ The `ErrorMessage` is displayed in a `ValidationSummary` control — provided you've added one to your page (as described later in this chapter in its own section, "ValidationSummary").



You can add as many validation controls to a page as you need to ensure quality data entry.

RangeValidator

The `RangeValidator` lets you check an entered value to verify that it falls within some range of acceptable values. It's the same idea as other controls in that you drag it to the page and set its `ControlToValidate` property to the name of the control you want validated. Then you set the `Maximum Value` and `Minimum Value` properties to define the range of acceptable values for the control.

RegularExpressionValidator

A *regular expression* is a symbolic way of defining complex validation criteria. You can do things like validate an entry against a pattern of characters, or

add complex “or” logic to a validation control. The language for creating regular expressions is extensive and beyond the scope of this book. But there are some ready-made ones that you can use without studying the whole language of regular expressions.



For an in-depth discussion of regular expressions, see <http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcomregularexpressions.asp>.

To validate a control against a regular expression, first drag a `RegularExpressionValidator` onto the form and set its `ControlToValidate` property to the name (ID) of the `TextBox` control you want to validate. Then click the `ValidationExpression` property, click the `Build` button, and you can scroll through a list of predefined standard expressions in the Regular Expression Editor (looking very regular in Figure 9-15).

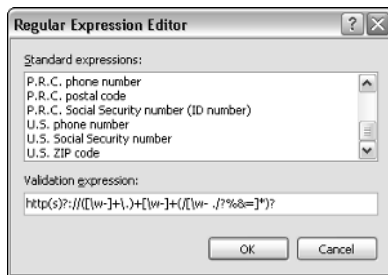


Figure 9-15:
The Regular
Expression
Editor.

If you find a standard expression you can use, click it and then click `OK`. In the Properties sheet, the expression appears as a bunch of parentheses, backslashes, and other weird characters. But that’s just what a regular expression looks like. It should work in your form anyway.

CompareValidator

Use the `CompareValidator` to check a text box entry by comparing it to a known value or the contents of another control. The most common use of this is when having a user enter a new password twice. Use a `CompareValidator` to compare the contents of the two controls, and reject both entries if they don’t match. Here’s how to use one:

1. Drag a `CompareValidator` control to your form and drop it there.
2. Check the new control by setting its `ControlToValidate` property to the name of the control.

3. Use the control to make a comparison of contents.

- To compare your control's contents to those of another control, set the `ControlToCompare` property to the name of the control to which you're comparing the current control.
- To compare the entry to some other known value, set the `ValueToCompare` property of the control to the comparison value.

CustomValidator

Use a `CustomValidator` if none of the other validation controls can do what you need done — and (oh yeah) you also happen to be a skilled programmer. If that's your thing, here are your steps:

- 1. Drag a `CustomValidator` control to the page.**
- 2. Set the control's `ControlToValidate` property to the name of the control you want to validate.**
- 3. Switch back to your form.**
- 4. Outside the Properties sheet, double-click the `CustomValidator` control you dropped onto the page.**

The code-behind file for the page opens with a new `CustomValidator1_ServerValidate` procedure all ready and waiting for you to type in your code.

- 5. Write your validation routine, close and save both files, then view the page in a Web browser to test it out.**

ValidationSummary

The `ValidationSummary` control displays a summary of all failed validations that occurred on the form. All you have to do is drop it on the form. You don't even need to change its properties.

In Design view, the `ValidationSummary` is just a placeholder showing no useful information. In a Web browser, it initially is invisible. After the user clicks the Submit button, it lists the `ErrorMessageText` of each control that failed validation. It's simple to use, and handy for users.

Using the Forms Designer

That about wraps it up for profile properties. However, because I've touched on the topic of forms here, it might be worth taking a moment to mention that putting form controls in tables isn't the only way to go in VWD. If you're accustomed to working with forms-design programs or graphics programs, it may seem awkward to have to use tables to create forms.

If you want to be able to stick controls anywhere on a page, you can: Use *absolute positioning*, the option that puts them where you say. You can use absolute positioning in any `.aspx` page, Master Page, Web User Control, or HTML page. The easiest way to do so is to start by creating or opening a page. Then, in Design view, follow these steps:

1. Choose **Layout** ⇨ **Position** ⇨ **AutoPosition Options** from the menu bar.
2. Make sure the first check box is selected.
3. Set the drop-down list option to **Absolutely Positioned**.
4. Optionally, enable snapping to pixels by choosing the second option, as shown in Figure 9-16.
5. Click **OK**.

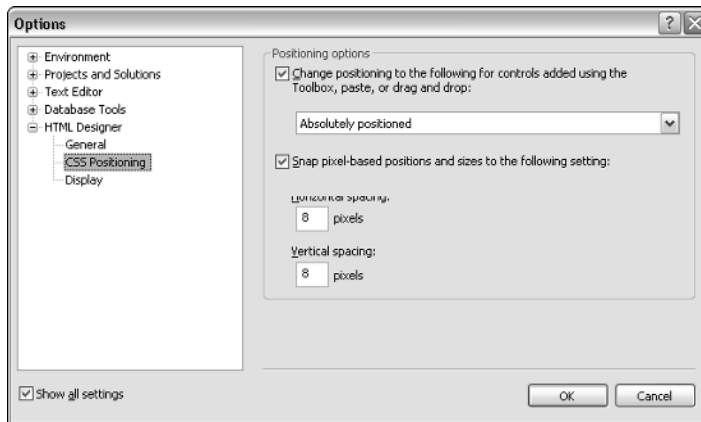


Figure 9-16:
Enabling
absolute
positioning.



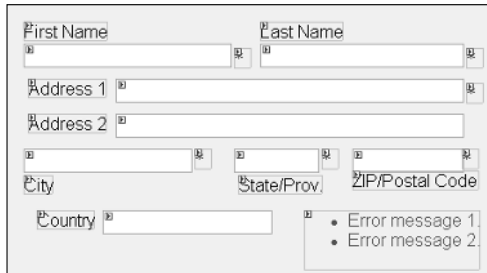
Choosing the option to snap to pixels is an easy way to line things up on the form. If you want total freedom in positioning things on the form (as you'd have in a graphics program), you can leave Snap to Pixels turned off. Feel free to try both settings to see how they differ and what works for you.

After you've enabled absolute positioning, any control you add to the page will initially jump to the upper-left corner of the page. But from there you can drag it anywhere you like; it'll stay wherever you drop it. You can arrange your controls however you see fit.

For each text box you intend to put on your page, you'll probably want to also add a `Label` control to hold some text describing what goes into the text box. To type text into a `Label` control, you have to change the label's `Text` property in the Properties sheet. If you double-click the label, you wind up on the code-behind page. (If you do that by accident, just close the code-behind page and choose `No` when asked about saving your changes.)

Figure 9-17 shows an example where I've absolutely positioned `Label`, `Textbox`, `Validation`, and `Button` controls on a page to create a form similar to `CreateProfile` or `EditProfile`. Because each item is absolutely positioned on the page, there's no need to use a table to get things lined up.

Figure 9-17:
Four controls positioned absolutely on a page.



The screenshot shows a web form with several text boxes and labels. The labels are 'First Name', 'Last Name', 'Address 1', 'Address 2', 'City', 'State/Prov.', 'ZIP/Postal Code', and 'Country'. There are also two error message boxes labeled 'Error message 1' and 'Error message 2'. The controls are arranged in a grid-like fashion, demonstrating absolute positioning.

Stacking absolutely-positioned objects

If you combine absolute and relative positioning, free-floating objects can cover (or be covered by) body text. With absolute positioning, you can set an object's Z-Index to define where you want it to appear in a stack. An item with a Z-Index less than 0 is placed beneath text, like the box labeled `Z = -1` in Figure 9-18.

Items with Z-Indexes greater than zero are stacked above text. For example, in Figure 9-18, the box labeled `Z = 1` is one layer above the text, so it covers text beneath it. The box labeled `Z = 2` is at an even higher layer, so it partially covers the boxes and text beneath it.

Setting the Z-Index of an absolutely-positioned item is pretty quick:

1. Right-click the item and choose `Style`.

Alternatively, you can click the item to select it, then click the `Style` property in the Properties sheet, and then click the `Build` button. Either way, the `Style Builder` opens.

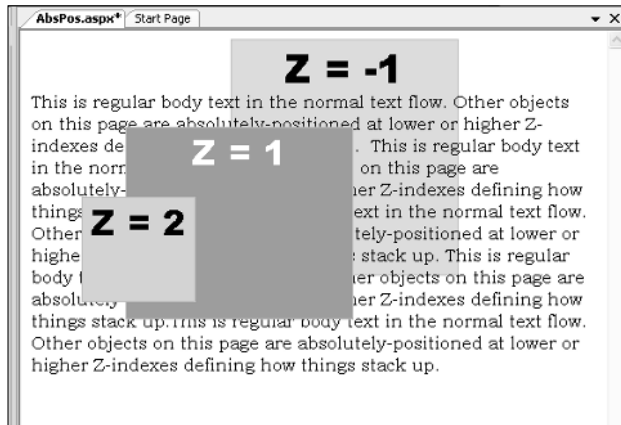


Figure 9-18:
High Z-Index items stack above low Z-Index items

2. In the Style Builder, click the Position category, use the Z-Index setting to set the item's Z value, and then click OK.

The on-screen object now has your chosen Z-Index.



To move an item quickly to the top or bottom of the stack without going through the Style Builder, click the item to select it. Then choose Format ⇨ Order ⇨ Send to Back (or Bring to Front, depending on where you want to put the item).

In Source view, absolutely-positioned items sport the style attributes that define the item's place on the page, as in this example:

```
style="position:absolute; top:152px; left:56px; z-index:2"
```

The `top` attribute defines the distance from the top of the page to the top of the control. The `left` attribute defines the distance from the left edge of the page to the left edge of the control. And of course `z-index` indicates the item's stacking position.



If you give an item a Z-Index less than zero, then can't select it in Design view, switch to Source view and make its Z-Index a positive number. Go back to Design view and make your changes. Then you can set the Z-Index back to a negative number.

Aligning absolutely-positioned objects

Before you align controls, get at least one control into position so you can use it as the *model* to which other controls will align. From there, it's just a couple of easy steps:

1. **Select all the controls you want to line up, saving the model control for last.**

Here's the sequence: Click the first item you want to align, hold down the Shift key while clicking others, and keep holding down the Shift key as you click the model. Be sure to select the model control last so it shows white sizing handles.

2. **Choose Format → Align from the menu bar and line things up the way you want.**

For example, in the left side of Figure 9-19, I selected a bunch of Label controls, making ZIP/Postal Code the model. The right side of that same figure shows the result of choosing Format → Align → Rights to get the right edges of all selected controls to line up to the right edge of the ZIP/Postal Code control.



It's easy to make mistakes while trying to get the hang of alignment options (trust me on this one). If you make a real mess of things, just press Ctrl+Z to undo the mess.

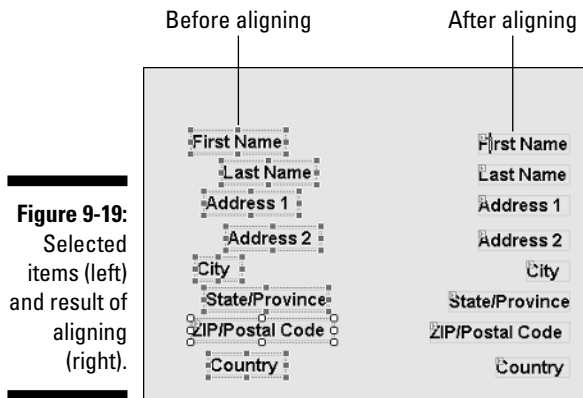


Figure 9-19:
Selected
items (left)
and result of
aligning
(right).

Sizing objects equally

If you want to make multiple objects the same size, the quick way is to pick or make a control to use as the model — and then follow these steps:

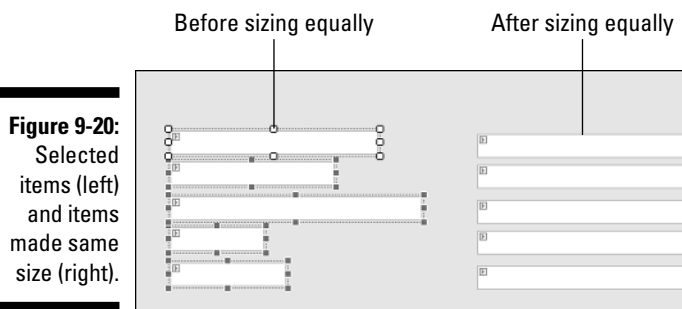
1. **Make a particular control the size and shape you want *all* the controls to be.**
2. **Select all the controls that you want to size.**

Make sure you select the model item last so it shows the white sizing handles. For example, in the left side of Figure 9-21, I selected the control at the top last, so it acts as the model.

3. Choose Format→Make Same Size from the menu bar.

4. Choose the uniform dimensions you want to apply to all controls.

You can make them the same width, the same height, or the same width and height. In the right side of Figure 9-20, I first made the controls all the same size (both height and width).



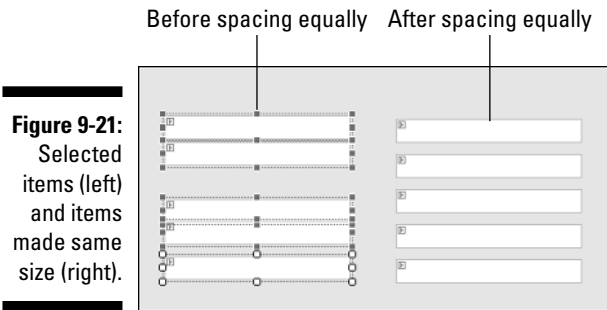
Spacing absolutely-positioned objects

If you want to equalize the spacing between multiple items in the page, first select all the items. No particular control plays the role of “model” here, so the order in which you selected the items isn’t important. After selecting the items, choose Format→Horizontal Spacing if you want to make the selected items equally spaced across the screen. Or choose Format→Vertical Spacing if you want to equally space the items down the screen. Note that you can also increase, decrease, or remove the spacing between items.



As always, if you really make a mess of things when trying to space multiple items, press Ctrl+Z to undo the change.

The left side of Figure 9-21 shows several `TextBox` controls selected. The right side shows those same selected controls after I chose Format→Vertical Spacing→Make Equal.



There's no rule that says you must use absolute positioning and all the various options for aligning, sizing, and spacing controls. Using a table often gets you the same result with a lot less effort.

Chapter 10

Using Themes

In This Chapter

- ▶ Creating themes for your site
 - ▶ Letting users choose their own themes
 - ▶ Applying themes to pages
 - ▶ Applying themes to Master Pages
-

In Windows and some other programs, users can choose a *theme*. The theme defines the general look and feel of the content in terms of color scheme, font, the way buttons are represented, and so forth.

If you're building a membership site where members are likely to spend a lot of time reading pages (for example, taking courses in an online school), then adding a similar theme capability to your Web site might be a good idea. Granted, it's a lot of work on your part. But it allows users to personalize the site content to their own needs and tastes, which, in turn, can make for a more pleasant experience at your site — and can encourage users to keep coming back for more — or even spread the word about your awesome site — instead of shopping elsewhere.

Creating Themes

The first step in adding themes to your site is to create an App_Themes folder and a single Theme subfolder. You might want to start by creating a default theme that everyone first sees when they open the site. To create an App_Themes folder and a default Theme folder, follow these steps:

- 1. Right-click the site name at the top of Solution Explorer and choose Add ASP.NET Folder⇨Theme.**
- 2. Type a name of your own choosing and press Enter.**

This is the name of your default Theme folder. I'll name my first sample theme `DefaultTheme`, but you can name yours anything you like.

In Solution Explorer, you now have a folder named `App_Themes`. It contains a subfolder with the name you entered in Step 2. The subfolder is empty; it's just a container in which you'll store the files that define the theme.

Creating Theme Folders

The whole idea behind themes is to offer the user several themes to choose from. Each theme you create will have its own subfolder within the `App_Themes` folder. So if you already have some ideas for themes you might create, you can create the Theme folders right now. To create a Theme folder, follow these steps:

1. **Right-click the `App_Themes` folder and choose `Add ASP.NET Folder` → `Theme`.**
2. **Type a name for the new theme and press `Enter`.**

In Figure 10-1, I've added a few theme folders named `Antique`, `ArtDeco`, `HeavyMetal`, `Nerdy`, `NoSquint`, and `Pastels`.

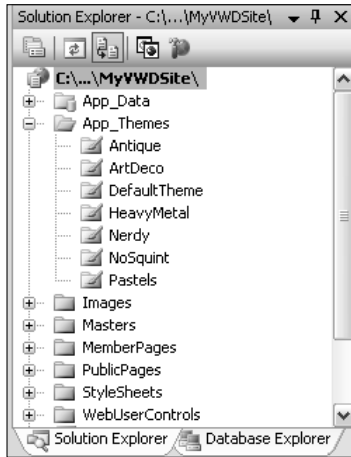


Figure 10-1:
New `App_Themes` folder and Theme folders.

So now you have a bunch of empty theme folders, which, so far, have no effect whatsoever on the look and feel of the Web site. Each folder is just a container for things to come.



You can rename and delete Theme folders like anything else. Just right-click and choose `Rename` or `Delete`.

What's in a Theme?

Depends on your point of view. From a user's perspective, a theme is the "look and feel" of the pages in a site. For you, the Web developer, a theme is a folder that contains files. Those files, in turn, define the look and feel of pages in the site. There are three types of files you can put into a Theme folder:

- ✓ **Pictures:** If a theme uses its own pictures for things like page backgrounds and icons, store those pictures in the Theme folder.
- ✓ **Cascading Style Sheets:** Use a standard Cascading Style Sheet to define things like font color, size, and other stylistic properties you define with the Style Builder.
- ✓ **Skins:** A *skin* defines the appearance of a type of control. For example, you can create a skin for all `TextBox` controls, another skin for all buttons, and yet another for all `TreeView` controls.

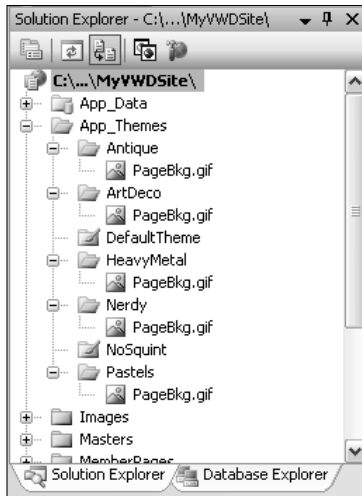
In a sense, themes are just a way of forcing you to get organized and stay organized. Because each theme has its own unique set of files, changing a theme is easy. You just open the theme folder and make your changes.

Using Pictures in Themes

With themes, you can use pictures to define things like the page background, the appearance of buttons and icons, and so forth. Any pictures that you intend to use as part of a theme need to be stored in the Theme folder. Assuming you already have the pictures, it's just a simple matter of dragging each picture's icon into the appropriate Theme folder in Solution Explorer.

In Figure 10-2, I created some page-background images for my site's Antique, ArtDeco, HeavyMetal, Nerdy, and Pastels themes. DefaultTheme and NoSquint just use plain white backgrounds, so no pictures are necessary. I dragged each image into its appropriate folder. Though it's not entirely necessary to do so, I renamed each image `PageBkg.gif`. I renamed them so each filename defines the picture's *role in the theme*, rather than the picture's content. But that's not something VWD requires you to do. You can name your own pictures however you like.

Figure 10-2:
Some
themes use
a page-
background
image.



Creating a Theme Style Sheet

To define the look and feel of pages in terms of font, text color, text size, background colors, border colors, and so forth, create a *theme style sheet*. A *theme style sheet* is just a normal Cascading Style Sheet stored within the theme folder. If you already have a style sheet, you can just copy that one into a theme folder. Then edit the copy that's in the theme folder to define what's different in this particular theme.

If you already have a Cascading Style Sheet that defines all the design elements of your site, you can keep that sheet and continue to use it in all your pages. In theme style sheets, you can define only those elements that are different from what the default style sheet specifies. That way you still have all your site-wide styles in one place, and themes define the styles of only those items that vary from the default style sheet.

Because of the way theme folders are organized, it makes more sense to name files by their role within the theme than by their content. So you might want to start by copying your default style sheet to a theme folder. Rename the copy in the theme folder to `StyleSheet.css`. Then double-click that style sheet to open and use as your starting point for creating the theme style sheet.

If you don't already have a style sheet for your site (or just prefer to start the theme style sheet from scratch), follow these steps:

- 1. Right-click the Theme folder to which you want to add a style sheet, and choose Add New Item.**
- 2. In the Add New Item dialog box, click Style Sheet.**

3. Click the Add button.

VWD adds a page named `StyleSheet.css` to the theme folder, and opens the page for editing.



Technically, you don't have to name the theme style sheet `StyleSheet.css`. But again, given the way theme folders are organized, it seems to make more sense to name files by their role in the theme rather than the specific file content.

Whether you create a style from scratch, or start with a copy of an existing sheet, you create and edit themes using the tools and techniques described in Chapter 6 to define your styles. To recap quickly:

- ✓ To create a new style, right-click some empty place in the style sheet, and choose **Add Style Rule**.
- ✓ To define a style using the Style Builder, right-click between the opening and closing curly braces of the style name, and choose **Build Style**.

After you create a `StyleSheet.css` page for one theme, and close it, you can then copy it to all your other theme folders to use as the starting point in designing those other themes.

The idea is to have a style sheet for each theme, as in the example shown in Figure 10-3. In that example, each `StyleSheet.css` page defines things like body text, font, and color; heading text font and colors; and so forth.

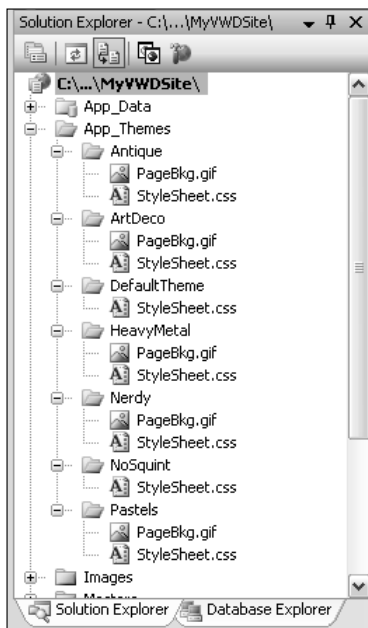


Figure 10-3: Each theme folder has a `StyleSheet.css` file

Excluding `DefaultTheme` and `NoSquint`, which don't use background pictures, each `StyleSheet.css` file defines a picture background. For example, the style rule for the `body` tag in the `Antique` theme's folder might look something like this:

```
body
{
    font-family: 'Bookman Old Style' , Serif;
    font-size: medium;
    color: #8b4513;
    background-image: url(App_Themes/Antique/PageBkg.gif);
}
```



You don't have to type the style rule manually, of course. You can use the Style Builder to define everything in every style sheet.

After you've defined pictures and style sheets for themes, you can take it a step further and define styles for ASP.NET controls within each theme. However, I don't refer to the styles for ASP.NET controls as *style sheets* — I call them *skins*.

Creating Skins

Exactly how far you want to take themes is up to you. You can design just about any page element using CSS styles and images. If you want to take it a step further and style ASP.NET controls like text boxes, buttons, and `TreeView` controls, you can do that, too. You use *skin* files to design ASP.NET controls.

A skin, like a style, is just the visual appearance of an item on a Web page. Most ASP.NET controls are *skinnable*, meaning you can define multiple styles for any given control. The way you define skins, though, is different from the way you define CSS styles.

Creating a skin file

The easiest way to create a skin is to start with a new instance of the type of control you want to style: a text box, for example, or a button. Then style that control using its Properties sheet or the Style Builder (or both). Then copy the resulting `<asp: ... > ... </asp: >` tags to a skin file, and remove any tag attributes that aren't stylistic and wouldn't be applied to all controls. For example, remove the `ID=` attribute because that's unique to each control you create and not relevant to the control's visual appearance.

Let's take it from the top and look at an example. First, create a Web Form page to use as an artist's canvas. Here's how.

1. Right-click the site name at the top of Solution Explorer and choose Add New Item.
2. Choose Web Form.
3. Deselect the “Use master page” check box because you don’t need one here.
4. Name the page anything you like.

I’ll use `Temp.aspx` for my page.

5. Click the Add button, and a new empty page opens up.

If the page opens in Source view, click the Design button to switch to Design view.

Next create an instance of the type of control you want to style. The following steps show you how:

1. In the interest of keeping it simple, drag a `TextBox` control from the Standard controls in the Toolbox onto the page.
2. To style the control, right-click it and choose Style (or choose Options from the control’s Properties sheet to define styles).

Figure 10-4 shows an example where I’ve right-clicked a `TextBox` control. You can also see the Properties sheet for the control.

3. View the page in a Web browser to see how the control will really look in a page. Then close the browser.

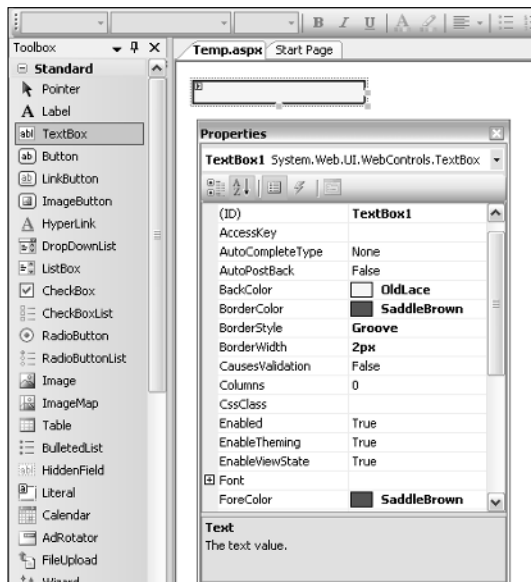


Figure 10-4:
A `TextBox`
control with
styled
Properties.

4. When you're happy with the way the control looks, switch to Source view.
5. Select the entire control, including the opening and closing `<asp:> . . . </asp:>` tags, nothing more, nothing less, as shown in the example in Figure 10-5.
6. Press **Ctrl+C** (or right-click and choose **Copy**) to copy the selected tag to the clipboard.

Figure 10-5:
A styled **Textbox** control's tag selected in Source view.

```

8 </head>
9 <body>
10 <form id="form1" runat="server">
11 <div>
12 <asp:TextBox ID="TextBox1" runat="server"
13     BackColor="OldLace"
14     ForeColor="SaddleBrown"
15     BorderColor="SaddleBrown"
16     BorderStyle="Groove"
17     BorderWidth="2px">
18 </asp:TextBox>
19 </div>
20 </form>
21 </body>
22 </html>
23

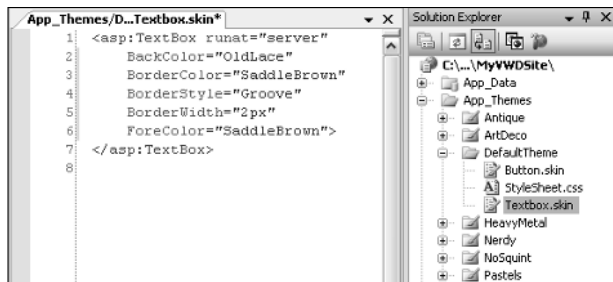
```

7. In **Solution Explorer**, right-click the theme file in which you want to create your skin file and choose **Add New Item**.
8. In the **Add New Item** window, choose **Skin File** and type a filename for the file.
I'd name the one I just created `Textbox.skin`.
9. Click the **Add** button.
10. Press **Ctrl+V** to paste the tags from the clipboard into the skin file.
There may be some comments (green text) in the skin file already. That's just a large comment that you can delete. It serves no purpose other than to tell you about naming skins.
11. Delete the `ID= . . .` attribute and control name (`TextBox1` in this example).

If you're designing a control that has other non-stylistic attributes in its tag, those should be eliminated to. For example, when designing a generic `TreeView` control, you wouldn't want to define a specific `Source` attribute in the skin. The skin is only about visual attributes.

Figure 10-6 shows how my `Textbox.skin` file looks after pasting in the tags, removing the green comment text, and removing the `ID="TextBox1"` attribute. As you can also see in the figure, this `Textbox.skin` file is in my `DefaultSkin` theme folder.

Figure 10-6:
Sample
Textbox.skin
skin file.



Default vs. named skins

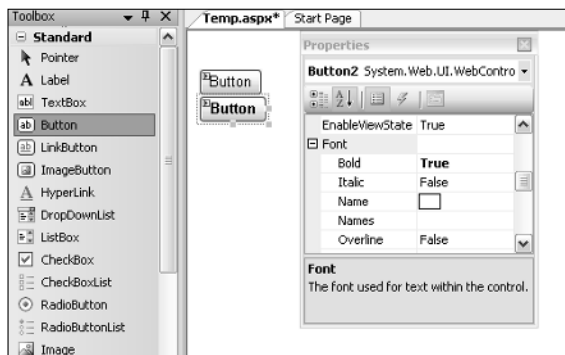
Any skin you create can be treated as either a *default skin* or a *named skin*. Here's the difference:

- ✓ **Default skins:** Apply automatically to all controls of the same type. For example, Textbox.skin would apply to all Textbox controls in all pages if treated as a default skin.
- ✓ **Named skins:** Apply only to controls in pages that specifically identify the skin by SkinID in their Properties sheet.

If you want to create a default skin, don't give the skin a SkinID attribute. The skin is applied to all controls that use the theme, except controls that request a specific skin by name. Most likely you'll want to create one default skin that applies to all instances of the control you've styled.

To define a specific skin that controls can use, give the skin a SkinID attribute. For example, suppose you want to have two styles of buttons in your site, a regular button as shown at the top of Figure 10-7, and a bold button like the one at the bottom of that same figure. In that figure, the bottom button is selected, and you can see where I've set its Bold property to True.

Figure 10-7:
Two Button
controls,
one
boldfaced.



Changing from Design view to Source view reveals the ASP tags for the two buttons, as shown in Figure 10-8. In the figure, I've already selected the tags for the two buttons, so I can copy them to a skin file.

Figure 10-8:
Tags for
buttons
selected.

```

5 <html xmlns="http://www.w3.org/1999/xhtml" >
6 <head runat="server">
7   <title>Untitled Page</title>
8 </head>
9 <body>
10 <form id="form1" runat="server">
11 <div>
12   <asp:Button ID="Button1" runat="server" Text="Button" />
13   <asp:Button ID="Button2" runat="server" Font-Bold="True" Text="Button" />
14 </div>
15 </form>
16 </body>
17 </html>

```

Next, just copy both tags to a new skin file. Because these are for ASP `Button` controls, you'd right-click a theme folder and choose Add New Item. Choose Skin File in the Add New Item dialog box, name the file `Button.skin` and click Add. Then paste the tags into the skin file.

Recall that you always need to remove from the skin file any attribute that will vary from one control to the next. That means you always have to take out the `ID=` attribute. In the case of a `Button` control, you also want to remove the `Text=` attribute because that defines the text that appears on the button. That will vary from one button to the next. (OK buttons, Cancel buttons, Submit buttons, and so forth, all have unique text.)

Finally, add a `SkinID="yourName"` attribute to all but the default skin. The *yourName* part can be anything you like. In the case of my bold button, I'd probably use something like `SkinID="Bold"`.

In my own `Button.skin` file I replaced the comment text with the `<asp:Button>` tags. I removed the `ID="name"` and `Text="Button"` attributes from both tags. Then I added a `SkinID="Bold"` to the second button style to differentiate it from the default button style, just above in the same skin file. The result is shown in Figure 10-9 where the `Button.skin` file in `DefaultTheme` contains a default skin for buttons, and a `Bold` skin for buttons.

Figure 10-9:
New
`Button.skin`
file in
Default-
Theme
theme.

The screenshot shows a code editor window titled 'App_Themes/De.../Button.skin*' with the following content:

```

1 <asp:Button runat="server" />
2
3 <asp:Button SkinID="Bold"
4   runat="server"
5   Font-Bold="True"
6 />
7

```

To the right, the Solution Explorer shows the project structure for 'C:\...MyWebSite\'. The 'App_Themes' folder is expanded to show 'DefaultTheme', which contains 'Button.skin', 'StyleSheet.css', and 'Textbox.skin'. Other theme folders like 'Antique', 'ArtDeco', 'HeavyMetal', and 'Nerdy' are also visible.

I've intentionally kept these skin examples small and simple, so as not to obscure the basic facts with lots of details. But you can style your controls however you see fit.

As with background pictures and style sheets, you'll need to create skin files for all your themes. (As you can imagine, one could end up putting a lot of time into this sort of thing.) The result would be as in Figure 10-10, which shows some of my sample theme folders. Though you can't tell from looking at the figure, the `StyleSheet.css`, `Textbox.skin`, and `Button.skin` files in each folder provide a style that's unique to that theme.

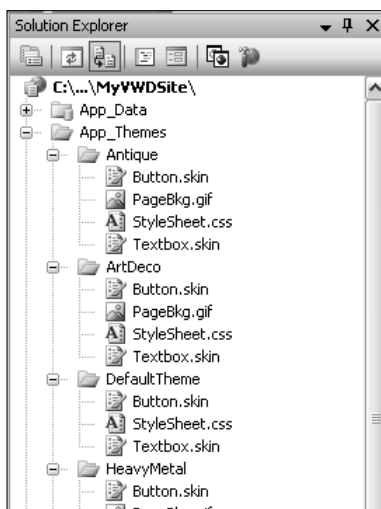


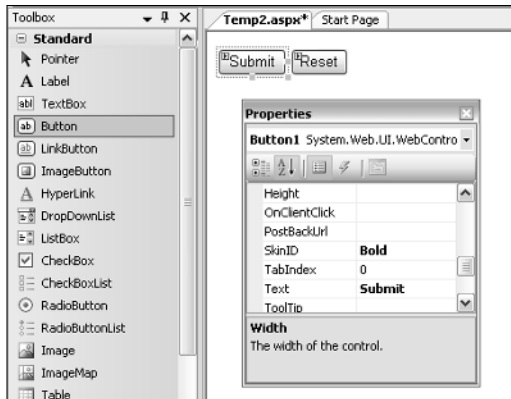
Figure 10-10: Each theme can contain its own skin files.

Using Themes in Pages

To take advantage of themes while creating and editing pages, you really don't have to do anything special; as always, style-sheet themes are applied automatically. Default skins are also applied automatically to ASP controls. For example, every `Textbox` control automatically takes on the appearance of the currently selected theme's `Textbox.skin` file.

If you want a control on a page to use a non-default skin file, just set the control's `SkinID` property to the name of the skin you want to use. For example, suppose you're creating a new page and you want to add some buttons to it. One of the buttons should use the `Bold` skin. After you add the button to your page, set its `SkinID` property to `Bold` (as in Figure 10-11).

Figure 10-11:
Using skins
in pages.



The control to which you apply a skin won't actually take on the skin's appearance immediately. (That's why the Submit button doesn't look any different from the Reset button in Figure 10-11.) Keep in mind that so far we've only taken the first step to adding themes to the site. There's still plenty more to do here.

There are many ways to implement themes. But it seems to me the most likely scenario would be that you want each member to be able to choose their own theme, which means to be able to store each user's theme preference as a profile property. So let's start with that.

Letting Members Choose a Theme

If the goal is to allow members to choose a theme, the first thing you'll want to do is create a profile property — say, named `PreferredTheme` — in which you store each user's preferred theme. To ensure that each new user has some theme selected, you can give the profile property the name of your default theme.

You can add the profile property to the `Web.config` file as described in Chapter 9. In my example where I already created profile properties, I would just add a new property named `PreferredTheme`, give it a default value of `DefaultTheme`, and a data type of `System.String` as shown in the following code:

```
<profile>
  <properties>
    <add name="FirstName" />
    <add name="LastName" />
    <add name="Address1" />
    <add name="Address2" />
    <add name="City" />
    <add name="StateProvince" />
    <add name="ZIPPostalCode" />
    <add name="Country" defaultValue="USA" />
    <add name="PreferredTheme" defaultValue="DefaultTheme"/>
  </properties>
</profile>
```

That takes care of having a place to store each user's preferred theme. In code, you can use `Profile.PreferredTheme` whenever you want to get, or set, the current user's preferred theme.

Next, you need some means of allowing users to explore themes and choose one, which means some sort of interactive form.

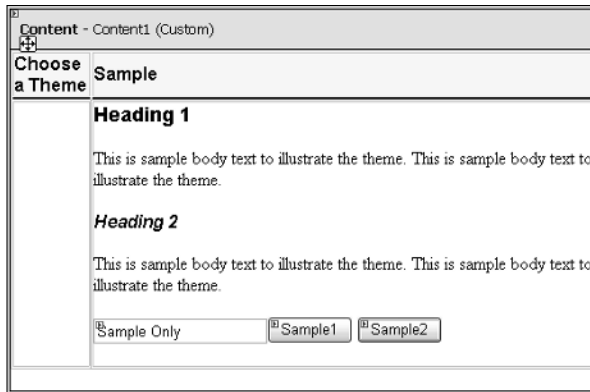
Creating a page for viewing themes

Next, you'd need a page where a user can go and take a look at various themes, and then choose a preferred theme. On my site, this page would go in my `MemberPages` folder because I'd only allow authenticated users to choose themes. So it's just a matter of right-clicking the `MemberPages` folder and choosing `Add New Item`. I'd choose `Web Form` as the template, enter `ChooseTheme.aspx` as my page, choose `Visual C#` as the language, and check both the check boxes for choosing a `Master Page` and using a `code-behind` page. Nothing new or different there.

To allow users to choose a theme, you'll need a control that allows users to choose any one of several mutually exclusive options — such as a drop-down list, a list box, or perhaps a set of option buttons (although you could just as easily use buttons or links).

The page also needs some content that uses styles defined in the themes. That way, when the user chooses a theme, you can simply apply the theme to show the user how the theme looks. How you design the page is entirely up to you, of course. As an example, I started off with the page shown in Figure 10-12.

Figure 10-12:
A page
that uses
elements
styled by
themes.



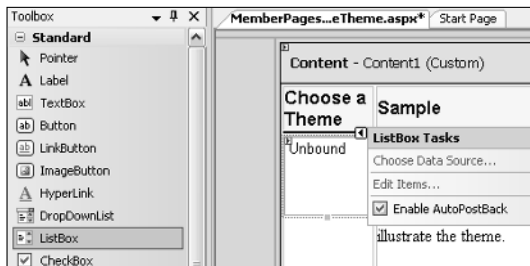
Okay, at this point there's nothing special about the page. It's just a table with some text, a Textbox control, and a couple of buttons that do nothing. (The `Sample2` button has its `SkinID` property set to `Bold` to test the `Bold` skin in my `Button.skin` themes.) The two top table cells have their `Class` properties set to `ColHead` to test the `TD.ColHead` style in my themes.

Creating a control for choosing a theme

Next comes the tricky part where I need a control that allows users to choose a theme. The trickiness comes from the fact that as soon as the user chooses a theme, we need to do a postback to the server to update the user's profile and also to apply the theme to the page. But let's start with the control itself.

I'll use a `ListBox` control for this example. Drag a `ListBox` control from the Standard controls in the Toolbox onto the page. Then choose `Enable AutoPostBack` from its common tasks menu (as in Figure 10-13), so a postback occurs as soon as the user chooses a theme.

Figure 10-13:
A `ListBox`
control
added to
the page.



Next you populate the `ListBox` with some options. Click Edit Items and use the `ListItem Collection Editor` (Figure 10-14) to list options to appear in the control. For each option, enter two things:

- ✔ **Text:** The text that appears in the box. This can be any text you want.
- ✔ **Value:** The value that the selection returns. This must exactly match the name of a Theme folder in the `App_Themes` folder.

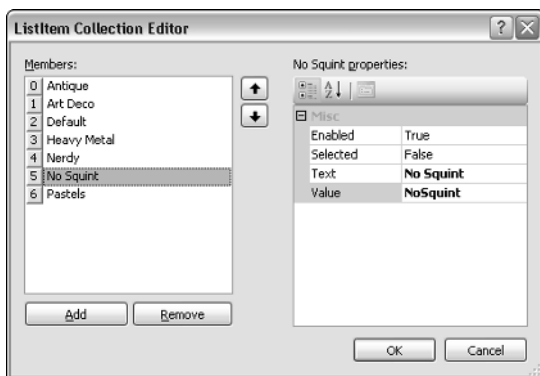


Figure 10-14:
The `ListItems`
Collection
Editor.

Each time you click the Add button, you're prompted to enter another item. In Figure 10-14, I've added an option for each theme. I left the highlighter on the No Squint option to illustrate that it's okay to show "No Squint" as two words in the list. However, the value of that option must be `NoSquint` to match the name of the Theme folder.

When you click OK in the `ListItems Collection Editor`, you'll be returned to the page and the options are visible. You can then set the height of the control by dragging its bottom edge (or by setting the `Height` property in the control's Properties dialog box).

Storing the preferred theme

When the user makes a selection from the `ListBox`, that control's `SelectedValue` property will equal the `Value` of the selected item. For starters, we need some code to copy the value to the user's `PreferredTheme` property. To get to the code-behind file, first double-click the `ListBox` control. The event handler for the `ListBox` is named `ListBox1_SelectedIndexChanged`.

The C# code needed to copy the selected value to the user's profile is simply this:

```
Profile.PreferredTheme = ListBox1.SelectedValue;
```

Another detail to think about here is the fact that when the user first opens the page, she will already have a theme name selected in her profile. Then the `ChooseTheme` page first opens; the `ListBox` control should show that current theme as the selected theme. So in the `Page_Load` event, you need some code to set the `SelectedValue` of the `ListBox` control to whatever is currently in the user's `PreferredTheme` property. Hence, the `Page_Load` event needs this statement:

```
ListBox1.SelectedValue = Profile.PreferredTheme;
```

But, as is often the case with page loads, we have to take postbacks into consideration. Every time the user clicks the `ListBox` control, that's going to cause a postback, as we really only want to load the preferred theme when the user first opens the page. So once again, we need some "if not postback" logic in the `Page_Load` event handler, as shown here (in C#):

```
if (!Page.IsPostBack)
{
    ListBox1.SelectedValue = Profile.PreferredTheme;
}
```

Applying a theme

Given our whole approach to themes in this chapter, it stands to reason that whenever a page loads, its theme must be set to the user's preferred theme. To get that done in your code, simply set `Page.Theme` to the user's preferred theme. In C#, the code looks like this:

```
Page.Theme = Profile.PreferredTheme;
```

But, you can't set a theme in the `Page_Load` event because the theme has to be applied before content and controls are rendered on the page. The `Page_Load` event fires after all that stuff is done, which is too late to apply a theme. So use the `Page_PreInit()` event handler instead.

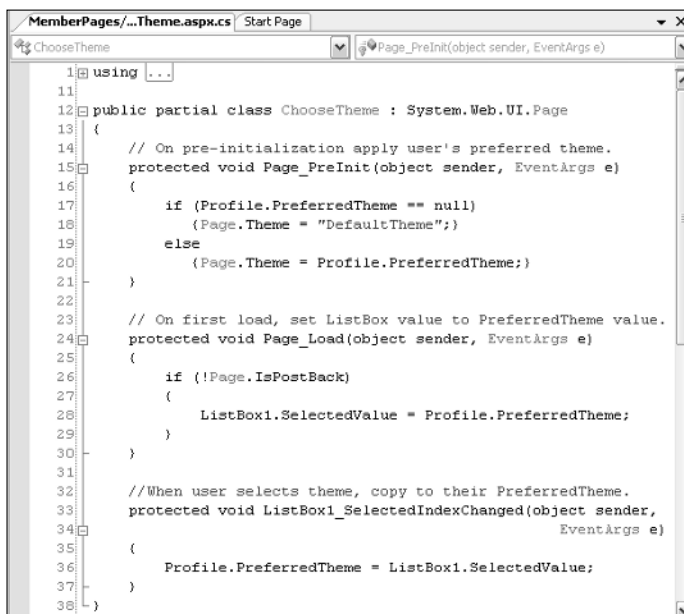
You won't necessarily find a predefined pre-init handler in the page. But you can just copy and paste one of the existing handlers into the code and change its name to `Page_PreInit`. Then type in the code to apply the preferred theme, as given here:

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Page.Theme = Profile.PreferredTheme;
}
```

In the above code, I'm assuming that every user will have something in their PreferredTheme profile, mainly because I gave that property a default value in Web.config. Just to play it safe, you could build a little logic into the code so if the PreferredTheme is "nothing," then DefaultTheme is used as the preferred theme, as given here (again in C#):

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (Profile.PreferredTheme == null)
    {
        Page.Theme = "DefaultTheme";
    }
    else
    {
        Page.Theme = Profile.PreferredTheme;
    }
}
```

The final C# code for the ChooseTheme.aspx page looks like Figure 10-15.



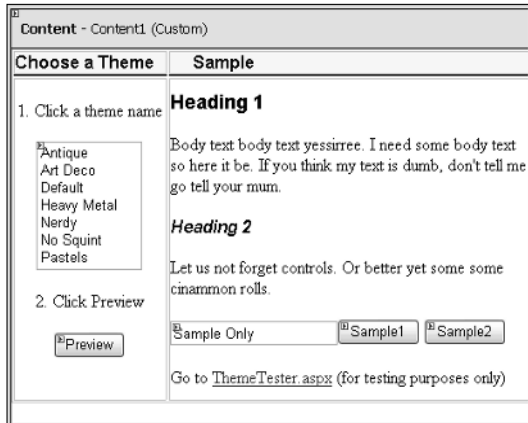
```
MemberPages/...Theme.aspx.cs Start Page
ChooseTheme Page_PreInit(object sender, EventArgs e)
1 using ...
11
12 public partial class ChooseTheme : System.Web.UI.Page
13 {
14     // On pre-initialization apply user's preferred theme.
15     protected void Page_PreInit(object sender, EventArgs e)
16     {
17         if (Profile.PreferredTheme == null)
18             {Page.Theme = "DefaultTheme";}
19         else
20             {Page.Theme = Profile.PreferredTheme;}
21     }
22
23     // On first load, set ListBox value to PreferredTheme value.
24     protected void Page_Load(object sender, EventArgs e)
25     {
26         if (!Page.IsPostBack)
27         {
28             ListBox1.SelectedValue = Profile.PreferredTheme;
29         }
30     }
31
32     //When user selects theme, copy to their PreferredTheme.
33     protected void ListBox1_SelectedIndexChanged(object sender,
34                                             EventArgs e)
35     {
36         Profile.PreferredTheme = ListBox1.SelectedValue;
37     }
38 }
```

Figure 10-15:
Here's the
C# code-
behind page
for Choose
Theme.
aspx.

All well and good — save for one little problem unique to this page: The PreInit event fires before the postback — which doesn't allow for the theme to be applied to the current page the first time you choose an option from the list box. To get around that, you can add a Preview button to the page beneath the ListBox, and provide some sort of instruction telling the user to click a theme name, then click Preview.

You don't have to program the button to do anything, because if you just drag a button from the Standard tools into the page, it will automatically do a postback when clicked later, in the browser. So basically you'd end up with something like Figure 10-16. The only control on that page that has an event handler is the `ListBox` control, which is handled by the `ListBox1_SelectedIndexChanged` shown in Figure 10-15.

Figure 10-16:
A Choose
Theme.
aspx
example.



A theme tester page

The `ChooseTheme` page simply allows a user to choose a theme. The theme won't actually be applied to any pages unless you specifically apply it. Then the theme of choice is the current user's preferred theme. So you need a `PreInit` handler on every page that will apply the theme.

To test it out, create a page named `ThemeTester.aspx`, and provide a link to it from the `ChooseTheme.aspx` page, as I did in Figure 10-16. In `ThemeTester` you can put any text or controls you want. But — most importantly — your code must apply the theme, like this:

```
protected void Page_PreInit(object sender, EventArgs e)
{
    if (Profile.PreferredTheme == null)
    { Page.Theme = "DefaultTheme"; }
    else
    { Page.Theme = Profile.PreferredTheme; }
}
```

That should be all that's required in the code-behind page of any page to which you want to apply user's preferred themes. After you get things working in `ChooseTheme.aspx` and `ThemeTester.aspx`, then it's just a matter of copying that same `Page_PreInit` handler to any page that needs to apply a user-selected theme.

Applying Themes to Master Pages

You can apply themes to Master Pages in same way you apply themes to an .aspx page. For example, you could allow users to choose background colors for the various table cells that make up a Master Page layout. Step 1 would be to open the Master Page and set the `Style` property of each table cell to a class name. Remove any explicitly-set `Style` properties that would conflict with the theme.

In Figure 10-17, for example, I removed the `Style` properties for the top table cell, and changed the `Class` property to a `MasterTop`. And though they're not visible in the figure, I removed the `Style` properties from the left column as well, and set its `Class` property to `MasterLeft`.

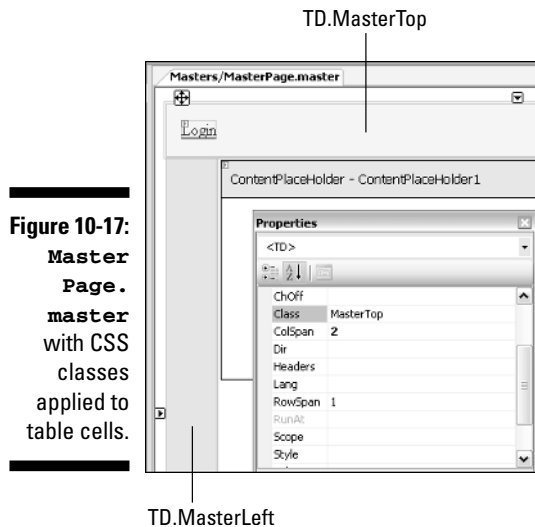


Figure 10-17:
Master Page.
master with CSS classes applied to table cells.

Close and save the Master Page. Then in your main style sheet (if you have one), set a default style for both elements as in the examples given here:

```
TD.MasterTop
{
    background-color: e0ffff;
    border-bottom: #483d8b thin solid;
}

TD.MasterLeft
{
    background-color: #fffff0;
    border-right: #483d8b thin solid;
}
```

You'll need the similar style rules in each theme folder's `StyleSheet.css` page, though the colors and other stylistic options you choose would vary from one theme folder to the next.

In the code-behind file for the Master Page (`MasterPage.master.cs` in my example), add `Page_PreInit` code to apply the user's preferred theme. The code is no different from the code you'd use in an `.aspx` page. So, in my example, it would look like this:

```
public partial class MasterPage : System.Web.UI.MasterPage
{
    // On pre-initialization apply user's preferred theme.
    protected void Page_PreInit(object sender, EventArgs e)
    {
        if (Profile.PreferredTheme == null)
        { Page.Theme = "DefaultTheme"; }
        else
        { Page.Theme = Profile.PreferredTheme; }
    }
}
```

Whether to allow users to customize Master Pages is really something you have to decide for yourself.



There's no rule that says a site can only have one Master Page. The fact is, a site can have as many Master Pages as you like.

As an alternative to allowing users to customize “the” Master Page for your site, you could have two Master pages. Use one fixed-and-unchanging Master Page for general pages. Then create a second Master Page for privileged content only, and allow users to set styles within that Master Page only.

Other Ways to Apply Themes

In all of the above examples of applying themes, you used the syntax `Page.theme=Profile.PreferredTheme` to apply the user's preferred theme to a page. Here's another syntax you can use as an alternative:

```
Page.StyleSheetTheme = Profile.PreferredTheme;
```

The difference between using `Page.Theme` and using `Page.StyleSheetTheme` is as follows:

- ✔ **Page.Theme:** The theme's style settings override any local style settings.
- ✔ **Page.StyleSheetTheme:** Behaves more like a Cascading Style Sheet — local style settings still take precedence over styles defined in themes.

Choosing one option or the other is largely a matter of personal taste. The advantage of using `Page.Theme` is that you know that whatever is defined in the theme “rules.” You won't get any unpleasant surprises when a control or a chunk of text ignores the theme because you previously set some style that's now overriding your theme.

The disadvantage to using `Page.Theme` is that if ever there was a need to override a theme-defined style for a particular control, there wouldn't be any way to do it, other than to create a special style rule for that one control.

In the long run, you're probably better off sticking to the `Page.Theme` approach and allowing themes to take precedence. You don't want to end up with hundreds of controls on hundreds of pages, all doing their own things and ignoring your themes. That can just make using (not to mention maintaining) the site all the more confusing and more laborious.

Defining a Site-Wide Default Theme

In the code samples above, themes are applied to pages *programmatically* in the sense that code defines which theme is used for any given page based on the user's `PreferredTheme` property. At a higher level, you can define a default theme to be used by all the pages in your site. You do so in the `Web.config` file for your site using either of the following syntaxes:

```
<pages theme="ThemeName" />
<pages styleSheetTheme ="ThemeName" />
```

where `ThemeName` is the name of a theme folder. For example, if you create a theme folder named `DefaultTheme`, and want to apply that theme to all pages in the site, you add the following line to your `Web.config` file:

```
<pages theme="DefaultTheme" />
```

Be sure to put the tag between the `<system.web>` and `</system.web>` tags in `Web.config` (as in Figure 10-18).



```
web.config*
<?xml version="1.0" encoding="utf-8" ?>
<system.web>
  <membership>
    <providers>
      <add type="System.Web.Security.SqlMembershipProvider"
          connectionStringName="LocalSqlServer"
          minRequiredPasswordLength="5"
          minRequiredNonalphanumericCharacters="0"
          passwordStrengthRegularExpression="" />
    </providers>
  </membership>
  <profile>
    <properties>
      <add name="FirstName"/>
      <add name="LastName"/>
      <add name="Address1"/>
      <add name="Address2"/>
      <add name="City"/>
      <add name="StateProvince"/>
      <add name="ZIPPostalCode"/>
      <add name="Country" defaultValue="USA"/>
      <add name="PreferredTheme" defaultValue="DefaultTheme"/>
    </properties>
  </profile>
  <pages theme="DefaultTheme" />
</system.web>
</configuration>
```

Figure 10-18:
Setting a default theme in Web.config.

The beauty of this approach is that every page opens with the default theme applied, instead of with no theme applied. As you create and review your pages, you know exactly where each item's style is coming from.



The default theme is *only* the default, in the sense that it's applied only when something else doesn't override it. Any theme that's applied programmatically will override the default theme.

Chapter 11

SQL Server Crash Course

In This Chapter

- ▶ Exploring database design
- ▶ Designing a database with tables and primary keys
- ▶ Creating your own SQL Server tables
- ▶ Linking tables

A big part of any dynamic data-driven Web site is the database that contains data for the site. Visual Web Developer allows you to use either Microsoft Access (.mdb files) or SQL Server 2005 (.mdf files) for storing data. Of the two, a SQL Server database provides better scalability and supports a greater number of simultaneous users. That's why it's the database most people will likely want to use (or would, if they knew what we know), so that's the one that gets the focus in this book.

As its title suggests, this chapter is a quick crash course in using SQL Server as the database for your Web site. You'll discover what SQL Server is, and how to create tables to store the data your Web site needs. You'll also get the lowdown on SQL, Structured Query Language, the tool you use to extract specific data from the database to display on Web pages. If you're already familiar with a database program like Microsoft Access, the most noticeable feature of SQL Server is that it has no user interface — no program window you can open from the Start menu. There's no table designer, no forms designer, nothing. The reason for that is simple: Microsoft Access is an *application program* that has to perform multiple tasks for a wide range of users — but SQL Server is a lot more specialized; it's just a *server*.

As a server, SQL Server 2005 is designed to provide data storage and Access to some other program rather than to a human who is sitting at the mouse and keyboard. The working interface for SQL Server isn't in its own separate program window. It's right in Visual Web Developer.

Crash Course in Database Design

Database design is an enormous topic that could easily fill several books the size of this one — and has — so I can't get into all the details here. But I can get you in the ballpark and clear up some of the important buzzwords that apply to all relational databases.

Tables, rows, and columns

The data (information) in a SQL Server database is organized into *tables*. Each table consists of *rows* (also called *records*) and *columns* (also called *fields*). Figure 11-1 illustrates the terms, using a sample table named `Items`. The field names (that is, the column names) are `ItemId`, `OurItemId`, `ItemName`, and `ItemPrice`. The table contains five rows (records), which happen to be numbered 10001, 10002, 10003, and so forth.

Rows (records) Columns (fields)

ItemId	OurItemId	ItemName	ItemPrice
10001	BWG-101	Introduction to Bird Watching	\$29.95
10002	BWG-201	Intermediate Bird Watching	\$29.95
10003	BWG-301	Advanced Bird Watching	\$39.95
10004	RWG-101	Introduction to Reptile Watching	\$29.95
10005	RWG-401	Recovering from Reptile Wounds	\$49.95

Figure 11-1:
A sample
database
table.

When you set up your site for Membership, VWD automatically creates several tables for you. To see those tables, first make sure your Web site is open in Visual Web Developer, then click the Database Explorer tab or choose `View` → Database Explorer from the menu bar.

To see the tables that are already in your database, expand the Data Connections, ASPNET.MDF, and Tables categories (if they're showing a + sign). Each item under the Tables heading represents a single table of data. The names of tables that VWD automatically creates all start with `aspnet_` as shown in Figure 11-2.

Each one of those tables contains rows and columns. To see the actual data that's stored in a table, right-click the table name and choose Show Table Data. For example, if you right-click `aspnet_Users` and choose Show Table Data, the `aspnet_Users` table opens. The table will contain one record for each user account you've created.

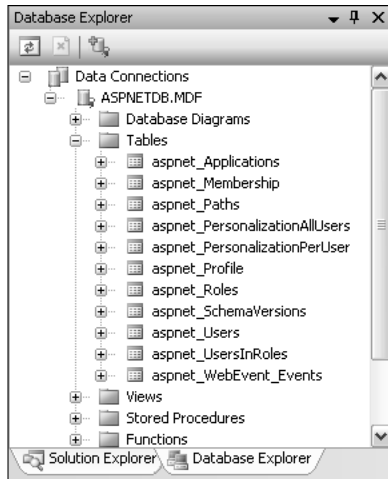


Figure 11-2:
Automatically
created
tables in a
Membership
site.

For example, by the time I got to this part of the book, I had created seven user accounts through the Web Site Administration Tool. Every account you create becomes a record in `aspnet_Users`. So opening my `aspnet_Users` table shows that it contains seven records, as shown in Figure 11-3.

aspnet_Users...ASPNETDB.MDF						
Appl...	UserId	UserName	Lowered...	MobileAlias	IsAnonymous	
1775...	bf93fd26-d11c...	Alice	alice	NULL	False	
1775...	f572a1ed-355...	Bob	bob	NULL	False	
1775...	66cd551e-470...	Carol	carol	NULL	False	
1775...	00a35f8e-34c...	Eeks	eeks	NULL	False	
1775...	8a116cb2-450...	Newbie	newbie	NULL	False	
1775...	9b5fc4e0-f29b...	TestMember	testmember	NULL	False	
1775...	14c650de-0b0...	Testy	testy	NULL	False	

Figure 11-3:
The
`aspnet_Users`
table data.



Don't add, change, or delete data in any of the `aspnet_` tables manually through Database Explorer. Always use the Web Site Administration Tool to manage membership data.

To close an open table, just click the Close (X) button in the upper-right corner, just as you'd close anything else that's open in the Design surface.

One-to-many, many-to-many

SQL Server 2005 is a *relational database-management system (RDBMS)* that not only stores data, but can also define multiple relationships among items

of data. It, and other products like it, exist mainly because in the real world, there are natural one-to-many relationships among different types of data. For example, suppose your site offers some kind of items to users. The items might be products you sell, courses you offer, Web seminars that people sign up for, or something like that.

In cases such as these there are two natural one-to-many relationships between users and the items being offered:

- ✓ Any *one* user might purchase *many* items.
- ✓ Any *one* item might be purchased by many users.

Any time you have two one-to-many relationships like that, you have what's called a *many-to-many* relationship. You have *many* users purchasing (or enrolling in, or attending) *many* items.

To illustrate, let's look at simplified versions of the `aspnet_Users` table and a table of items that the site offers to users. At the left side of Figure 11-4 is a table showing a couple of columns and some rows from a table of users. Each record in that table represents a single user account.

Figure 11-4:
Sample
`aspnet_Users`
and
`Items`
tables.

How do I
connect
these?

UserId	UserName
bf93f...	Alice
f572...	Bob
66cd...	Carol
00a3...	Eeks
8a11...	Newbie
9b6fc...	TestMember
14c6...	Testy

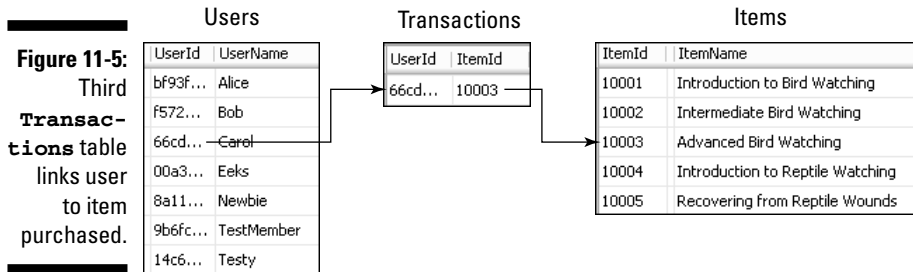
ItemId	ItemName
10001	Introduction to Bird Watching
10002	Intermediate Bird Watching
10003	Advanced Bird Watching
10004	Introduction to Reptile Watching
10005	Recovering from Reptile Wounds

On the right side is a simple `Items` table. Each record in that table represents a single item offered to users. In between the two tables is the burning question “How do I connect these?” The answer is pretty straightforward: “By creating a third table that keeps track of who purchased what.”

It might help to think of it this way. Each time a user purchases a product, that's a transaction. You need to keep track of these transactions — specifically, who purchased what — and that requires two pieces of information per transaction: *who* (a `UserId`), and *what* (an `ItemId`).

So, at the very least, this third table (which I'll name `Transactions`) must contain two fields: One field to record the user's identification (`UserId`), and the other to record the item purchased (`ItemId`). You can imagine this as a table with two fields named `UserId` and `ItemId` between the `aspnet_Users` table and the `Items` table.

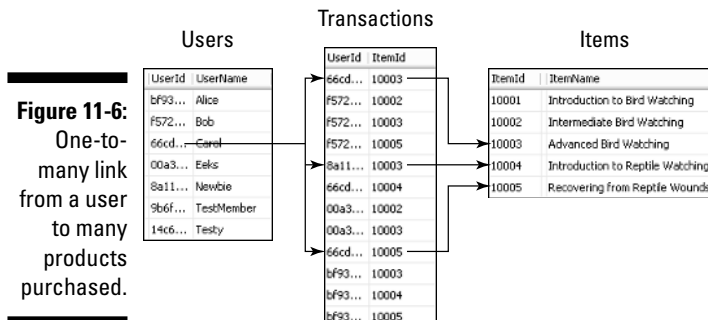
Now, imagine that user Carol (UserId 66cd...) purchases item 10003. The Transactions would need a new record with 66cd in its UserId field and the 10003 in its ItemId field. If you then trace an imaginary line from the user to the transaction to the item, you'll see how the record in the Transactions table provides the link from a specific user to a specific item purchased, as in Figure 11-5.



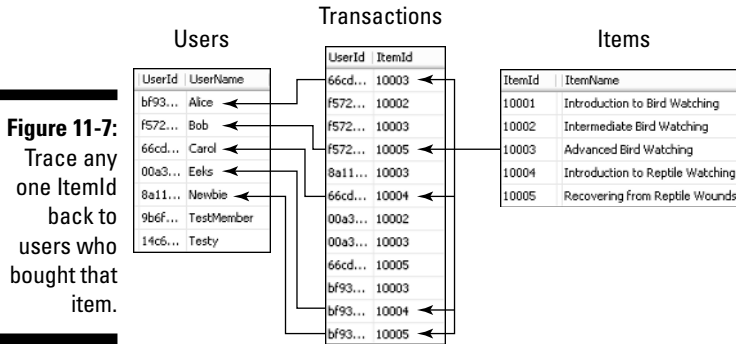
Every time a user purchases something, the Transactions table grows by one record. As time goes by, the Transactions table continues to grow, perhaps to thousands of records. Each record represents a single transaction where a specific user purchased a specific product.

Recall that the natural many-to-many relationship between users and items is actually two one-to-many relationships. Any one user might purchase many items, and any one item might be purchased by many users. The Transactions table provides the “map” that allows code to find all transactions by any one user.

No matter how many records are in the Transactions table, that table still provides the one-to-many link from users to items. To illustrate, pick one user, such as Carol (UserId 66cd...). Each record in the Transactions table that has 66cd... is a transaction made by that user. The ItemId field in Transactions identifies exactly which item the user purchased, as shown in Figure 11-6.



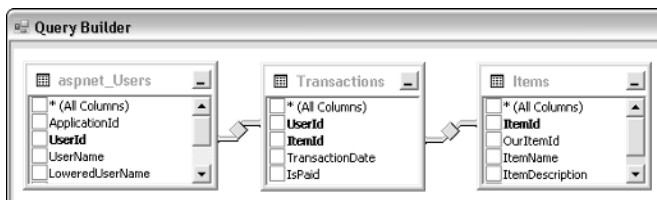
There's also a one-to-many relationship between items and users. The `Transactions` table, once again, provides the map describing which users purchased an item. Pick any one item, say 10003, from the `Items` table. Find records in the `Transactions` table that have 10003 in their `ItemId` field, and you have a link back to each user that purchased that product, as shown in Figure 11-7.



In real life, of course, you don't draw lines between records in tables to get information. In fact, you don't look at the tables at all — instead, you create queries to get information.

However, while I'm on the subject of lines that connect things between tables, any time you extract data from all three of the tables (users, transactions, and items), your query must contain fields of all three tables. And the primary and foreign keys that link the tables must be connected by join lines in that query, as shown in Figure 11-8. We'll discuss the roles of the primary and foreign keys in detail as we progress through the chapter.

Figure 11-8:
A query's view of connecting lines between tables.



So that's how database design works, in a conceptual sense. When there is a natural many-to-many relationship between items in two separate tables, you need a third table that contains records stating who purchased what. That third table provides the many-to-many link needed to extract meaningful data from the tables.

However, if you were to try this technique right now — using only what’s covered in the book up to this point — there’s a good chance yours wouldn’t work. That’s because there are rules to follow to get this sort of thing to work — two in particular:

- ✓ Every table on the “one” side of a one-to-many relation must have a primary key, a field that uniquely identifies each record in that table. In this example, both the users and items tables must have a primary key.
- ✓ The transaction table must contain at least two fields whose names and data types match the names and data types of the two tables on the “one” side of the relationships.

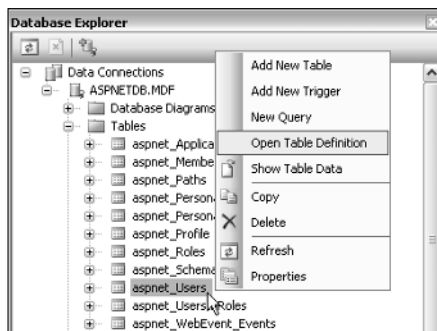
To understand and apply those rules, the first order of business is to understand data types and primary keys. Let’s start with data types.

SQL Server Tables

Every column in every table has a *data type* that defines the type of data stored in the column. To view each column’s data type, open the *table definition*, rather than the table data. When you open a table definition, you don’t see any of the actual data that’s stored in the table. Instead, you see the *structure* of the table. The table’s structure shows the name, data type, and whether the field allows nulls.

To see an existing table’s structure, right-click the table’s name in Database Explorer and choose Open Table Definition, as shown in Figure 11-9.

Figure 11-9:
How to view
a table’s
structure
(definition).



For example, if you right-click the `aspnet_Users` table in Database Explorer and choose Open Table Definition, you see field names listed down the left side of the grid that opens. Each of those names represents a column that appears at the top of the `aspnet_Users` table when you’re viewing the table’s data. To the right of each field name is the Data Type of that field, as shown in Figure 11-10.

Figure 11-10:
Table
definition
(structure)
of the
aspnet_
Users
table.

Column Name	Data Type	Allow Nulls
ApplicationId	uniqueidentifier	<input type="checkbox"/>
UserId	uniqueidentifier	<input type="checkbox"/>
UserName	nvarchar(256)	<input type="checkbox"/>
LoweredUserName	nvarchar(256)	<input type="checkbox"/>
MobileAlias	nvarchar(16)	<input checked="" type="checkbox"/>
IsAnonymous	bit	<input type="checkbox"/>
LastActivityDate	datetime	<input type="checkbox"/>

Here's what the table definition for the built-in `aspnet_Users` table tells you about columns in that table:

- ✔ There are seven fields (columns) in this table, their names are listed in the Column Name column (`ApplicationId`, `UserId`, `UserName`, and so forth).
- ✔ The data type of each field is visible just to the right of the field name. For example, both the `ApplicationId` and `UserId` columns are of the `uniqueidentifier` data type.
- ✔ The `UserId` field is the primary key for this table, as indicated by the key symbol to the left of its name.
- ✔ When entering new records into this table, it's okay to leave the `MobileAlias` field empty (because its `Allow Nulls` check box is selected). But no other fields in the record can be left blank.



Never change or remove anything in any `aspnet_` table's structure. Those tables are created and used by the membership system and any changes you make could cause the whole membership system to stop working.

When you're creating your own tables, it's important to choose data types wisely. A field's data type defines what you can put into the field — and what you can do with the information stored in that field. The main data types available to you in SQL Server 2005 Express are categorized as follows:

- ✔ **Text:** Character data like the text you're reading right now, people's names, product names, and so forth.
- ✔ **Number:** Also called *scalar values*, these are real numbers like quantities and dollar amounts on which you can do math.
- ✔ **Date/Time:** Dates and time of day.
- ✔ **Boolean:** A value that can be only `True` or `False`.
- ✔ **Binary:** Pictures and other kinds of data that aren't text, numbers, or dates.
- ✔ **Other:** Specialized data types such as `uniqueidentifier`, `xml`, `timestamp`, and `sql_variant`.

The most common type of information stored in tables is text, so we'll look at the text data types first.

Data types for storing text

In the computer biz, a chunk of text is referred to as a *string*, shorthand for “a string of characters.” The following are all examples of strings:

- ✓ Banana
- ✓ Andy Adams
- ✓ 123 Oak Tree Lane
- ✓ Hello world, how ya doin' today?
- ✓ (215) 555-1234 (phone number)
- ✓ 123-45-6789 (Social Security number)
- ✓ 00453-4321 (ZIP Code)



Phone numbers, social security numbers, and ZIP codes are not true numbers (scalar values) that you'd ever add, subtract, multiply, or divide. So they must be stored as strings (usually the `char` data type) rather than as numbers. If you use a number data type for any of those fields, you won't be able to use leading zeroes in Zip codes (like 01234), or use parentheses and hyphens in phone numbers, because such things are not allowed in scalar values.

Text comes in two basic flavors:

- ✓ **Unicode text:** Requires two bytes (16 bits) per character and can include characters from virtually any human language.
- ✓ **Non-Unicode text:** Uses only one byte (eight bits) per character, but is limited to characters in the English alphabet.

You always want to be efficient in how you store data. So if a field should contain only English characters (A–Z) and such, a non-Unicode data type would be most efficient. But, given that we live in a Web-connected world, you may often need characters from other languages, in which case you'd use a Unicode data type — and storing the data would cost you twice as much disk space.

When defining a field to store either type of text, you can choose between fixed-length and varying-length. Fixed-length is more efficient when all the values stored in the field are the same length, or very similar in length. For example, if you were to create a field to store product identifiers, and all the product identifiers follow a similar pattern — say, `ABC-123` — then a non-Unicode fixed-length text field would be ideal. In the `ABC-123` example, a fixed length of 7 characters would do the trick (the hyphen counts as a character).

Often, there's no way to predict how much text a field will store. For example, if the field is storing text that people have typed into a form, there's no telling

exactly how many characters a given user might type. You would want to use a variable-length text field for that sort of thing. The amount of storage space needed to store each string would be roughly equal to the length of each string.

Given that there are two types of text (Unicode and non-Unicode), and two ways to define string length (fixed-length or variable-length), there are four main data types for storing text data. The length of string allowed in the field — represented by n in the list that follows — determines how much storage space is required to store data in the column:

- ✓ **char(n)**: Fixed-length non-Unicode text where n defines the actual length of each string.
- ✓ **nchar(n)**: Fixed-length Unicode text, where n defines the actual length of each string.
- ✓ **varchar(n)**: Variable-length non-Unicode text where n defines the maximum length of a string stored in the field, up to 8,000 characters.
- ✓ **nvarchar(n)**: Variable-length Unicode text, where n defines the maximum length of each string, up to 4,000 characters.

Notice the pattern of the data type names. The names of data types that store Unicode data start with the letter n (for *national*), as in `nchar` and `nvarchar`. Data types that store variable-length strings all contain *var*, as in `varchar` and `nvarchar`.



For truly enormous chunks of text, SQL Server offers `varchar(MAX)` and `nvarchar(MAX)`, which can store strings containing up to two billion characters. But let's stick with the data types most commonly used in Web sites for now.

Number data types

In the computer world, the term *number* usually refers to actual *scalar values*, the kinds of numbers on which you can do math. That includes things like quantities and dollar amounts. As mentioned, it doesn't refer to types of data we casually refer to as numbers in day-to-day speech (such as phone numbers, Social Security numbers, or ZIP codes).

Scalar values come in two basic flavors:

- ✓ **Integers**: These are whole numbers with no decimal point.
- ✓ **Floating-point numbers**: These types of scalar values can contain a decimal point as in 0.10 or 123.45 or 12,345,678.987654321. There are also currency data types specifically used for storing dollar amounts like \$99.99.

Starting with integers, there are four data types to choose from. Each provides for a range of acceptable numbers, and each has its own storage requirements. You want to choose the smallest one you can, but make sure it's not too small to store the kinds of numbers you'll need.

For example, if the field is going to store a number between 1 and 10, then the `tinyint` data type will do just fine, because it can store numbers from 0 to 255. If the field will be storing much larger numbers — perhaps up to two billion (or some other value with a long string of zeros) — use the `int` data type. Here are the integer data types and the range of values each can store:

- ✔ **`tinyint`**: From 0 to 255
- ✔ **`smallint`**: From -32,768 to 32,767
- ✔ **`int`**: From -2,147,483,648 to 2,147,483,647
- ✔ **`bigint`**: From -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

For storing dollar amounts, your best bet is to use the `smallmoney` or `money` data types. For unit prices, you could probably get away with the `smallmoney` data type, unless some of your products cost more than \$200,000.00. For larger monetary values, use the `money` data type. Here is the range of values each data type can handle:

- ✔ **`smallmoney`**: From about -\$214,748.00 to about \$214,748.00.
- ✔ **`money`**: From about -\$922,337,203,685,477.00 to about \$922,337,203,685,477.00.

Floating-point numbers are for really big or precise numbers used in math and science, probably not the kind of thing you'd use much in a Web site — unless, of course, the Web site is about math or science. Several data types are available for storing floating-point numbers. I imagine the most commonly used would be the `decimal` data type.

With the `decimal` data type you can define a numeric value in terms of both precision and scale. The *precision* defines the total number of numeric digits in the number, the *scale* defines the number of digits to the right of the decimal point. For example, the number 1234.5678 has a precision of eight (because it contains 8 numeric characters) and a scale of 4, because there are four numbers to the right of the decimal point.

The maximum precision is 38 digits. The scale can be anywhere from zero (no digits to the right of the decimal point) to whatever the precision value is set to (all numbers to the right of the decimal point). For example, a field with its data type set to `decimal(38,18)` could contain 38 numeric digits, with 18 of them to the right of the decimal point. So a field could store a number like this:

```
99,999,999,999,999,999,999.999999999999999999
```

Other data types that can store floating-point numbers include `numeric`, `float`, and `real`. (Their ranges are shown in Table 11-1 later in this chapter.) It seems unlikely you'd use those in a Web site, so let's just keep forging ahead with the more likely data types here.

Boolean (bit) data type

A Boolean field is one that can contain only one of three possible values, Null, 0 (False), or 1 (True). In SQL Server, you use the `bit` data type to define a Boolean field. Back in Figure 11-10, the `IsAnonymous` field is the `bit` data type because it can contain only two possible values. A user is either anonymous (True) or isn't anonymous (False).



The `bit` data type is the same as the Yes/No data type in Microsoft Access, in case you happen to be familiar with that product.

Data types for date and time

There are two main data types for storing dates, `datetime` and `smalldatetime`. The `smalldatetime` data type will usually do the trick as it can store dates from January 1, 1900 to June 6, 2079. If your database needs to store dates outside that range, you can use the `datetime` data type, which can handle dates from January 1, 1753 to December 31, 9999.

The uniqueidentifier data type

The `uniqueidentifier` data type defines a field that can store a GUID (Globally Unique Identifier). A GUID is a long string of characters that looks something like this:

```
8a116cb2-4503-4f83-870e-4fa50bee923a
```

The tables that Visual Web Developer creates for membership use the `uniqueidentifier` data type for the `ApplicationID` and `UserID` fields. For example, if you right-click `aspnet_Users` in Database Explorer and choose Show Table Data, you'll see a record for each user account you've created so far. If you widen the first two columns enough to see the full contents of each, you'll see that each contains a GUID, as shown in Figure 11-11.

Figure 11-11:
ApplicationId and UserId are both unique identifier fields.

ApplicationId	User Id	UserName	LoweredUserName
17751c60-f3fa-4a0e-b471-d5263e321df4	bf93fd26-d11c-446c-b065-fd1de82182dc	Alice	alice
17751c60-f3fa-4a0e-b471-d5263e321df4	f572a1ed-3553-40c7-99ce-1ad3bcece831	Bob	bob
17751c60-f3fa-4a0e-b471-d5263e321df4	66cd551e-470c-4e57-9de3-d197bb6abc98	Carol	carol
17751c60-f3fa-4a0e-b471-d5263e321df4	00a35f8e-34ce-4fb1-b7b4-8f2792e9b799	Eeks	eeks
17751c60-f3fa-4a0e-b471-d5263e321df4	8a116cb2-4503-4f83-870e-4fa50bee946a	Newbie	newbie
17751c60-f3fa-4a0e-b471-d5263e321df4	9b6fc4e0-f29b-4fef-b966-d349222ce8f2	TestMember	testmember
17751c60-f3fa-4a0e-b471-d5263e321df4	14c650de-0b04-4562-b255-586163ae62cb	Testy	testy
NULL	NULL	NULL	NULL

You may be wondering where all the weird GUIDs came from. The `ApplicationId` identifies your entire Web site, and is automatically created when you first set up membership for your Web site. It's the same for each user because

each user is a member of the current Web site. The `UserId` is automatically created each time you create a new user account through the Web Site Administration Tool or the `CreateUserWizard` control.

So what does the GUID value *mean*? Absolutely nothing. And that’s the whole point. It’s a randomly-created value that has no meaning. But it’s still a good thing because it’s guaranteed to be unique, and uniqueness is a critical factor in fields that might be used as primary keys. Think of it as being like a person’s Social Security number. Everyone has one, but no two people have the same one. The Social Security number has no “meaning” — you just take whatever Social Security number you get handed.



Actually, your Social Security number *is* a primary key in the government’s databases. Your Social Security number uniquely identifies you among all the millions of people in the country, because nobody else has the same Social Security number that you do.

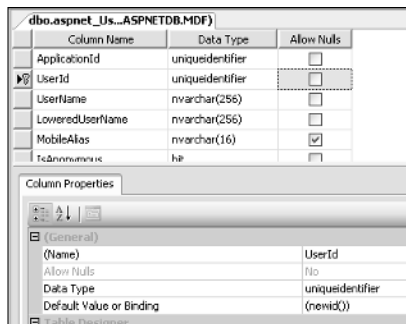
The `ApplicationId` is a unique identifier for your Web site. In the `aspnet_Users` table, every record has the same `ApplicationId` value in its field. That’s because each of these users is a member of the same *application* (which means *Web site* in this case).

The `UserId` field is the random GUID assigned to each user automatically when you (or a user) creates a user account. No two users will ever have the same `UserId`. And once entered, a user’s `UserId` never changes. Even users who change their own `UserNames` will still belong to the same user accounts they initially created — because the `UserId` value won’t change.

Assigning GUIDs automatically

Every time you or someone else creates a new user account, that new account automatically is assigned a GUID. The reason has to do with the `Default Data Value or Binding` column property for the field. If you click on the `UserId` field name in the table definition, then look at that property, you’ll see it contains `(newid())` as shown in Figure 11-12.

Figure 11-12:
The `UserId` field shows `(newid())` in its **Default Value or Binding** property.



That `newid()` property is what makes that field in that record receive a value automatically each time a record is created. If you looked at the `DefaultValue` or `Data Binding` property for the `ApplicationId` field, you would not see a `newid()` there. That's because the `ApplicationId` field does not get a new, random value each time you create a record. The value in that field is always the same, the GUID that specifically identifies your Web site.



In `aspnet_Users`, the `ApplicationId` field provides a link to the `aspnet_Applications` table. In `aspnet_Applications`, the `ApplicationId` field is a primary key.

The data types I've discussed so far don't represent the full set of SQL Server data types, just the ones you're most likely to use in your Web sites. For future reference, Table 11-1 summarizes all these data types. For more detailed information on data types, a heftier list of them, or any other facet of SQL Server, refer to SQL Server Books Online or a book that's strictly about SQL Server 2005.

<i>Data Type</i>	<i>Used to Store</i>	<i>Space Consumed</i>
<code>bigint</code>	Integers from -2^{63} ($-9,223,372,036,854,775,808$) to $2^{63} - 1$ ($9,223,372,036,854,775,807$)	8 bytes
<code>binary(n)</code>	Fixed-length binary data, where n can be any value from 1 to 8,000	n bytes
<code>bit</code>	1, 0, or Null	1 byte per 8 bit fields
<code>char(n)</code>	Fixed-length text, where n can be from 1 to 8,000	n bytes
<code>datetime</code>	Dates from January 1, 1753, through December 31, 9999, to 3.33 milliseconds of accuracy	8 bytes
<code>decimal(p, s)</code>	Numbers from $-10^{38} + 1$ to $10^{38} - 1$	Depends on p
<code>float(n)</code>	Numbers from $-1.79E + 38$ to $-2.23E - 38$, 0, and $2.23E - 38$ to $1.79E + 38$	Depends on n
<code>image</code>	Variable-length binary data to $2^{31} - 1$ (2,147,483,647) bytes	$length + 2$ bytes

Data Type	Used to Store	Space Consumed
int	Integers from -2^{31} ($-2,147,483,648$) to $2^{31} - 1$ ($2,147,483,647$)	4 bytes
money	Dollar amount from $-922,337,203,685,477.5808$ to $922,337,203,685,477.5807$	8 bytes
nchar (n)	Fixed-length Unicode text to 4,000 characters	$2 * \text{length}$
ntext	Variable-length Unicode data or binary data to $2^{30} - 1$ bytes	$2 * \text{length}$ bytes
numeric (p, s)	Numbers from $-10^{38} + 1$ to $10^{38} - 1$	Depends on p
nvarchar (n)	Variable-length Unicode text up to 4,000 characters	$2 * \text{length} + 2$ bytes
nvarchar (MAX)	Variable-length Unicode data to $2^{31} - 2$ bytes	$2 * \text{length} + 2$ bytes
real	$-1.18E - 38$, 0 and $1.18E - 38$ to $3.40E + 38$	4 bytes
smalldatetime	Dates from January 1, 1900, through June 6, 2079, accurate to 1 minute	4 bytes
smallint	Integers from -2^{15} ($-32,768$) to $2^{15} - 1$ ($32,767$)	2 bytes
smallmoney	Dollar amounts from $-214,748.3648$ to $214,748.3647$	4 bytes
sql_variant	Non-specific data type up to 8,016 bytes	Depends on data stored
text	Variable-length text to $2^{31} - 1$ ($2,147,483,647$ characters)	Depends on code page of server
timestamp	Time of last change to row	8 bytes
tinyint	Integers from 0 to 255	1 byte

(continued)

Table 11-1 (continued)		
Data Type	Used to Store	Space Consumed
unique-identifier	GUIDs (Globally Unique Identifiers)	16 bytes
Varbinary(<i>n</i>)	Variable-length binary data, where <i>n</i> can be 1 to 8,000	<i>length</i> +2 bytes
varbinary (MAX)	Variable-length data up to $2^{31} - 1$ bytes in length	<i>length</i> + 2 bytes
varchar(<i>n</i>)	Variable-length text where <i>n</i> can be from 1 to 8,000	<i>n</i> +2 bytes
varchar (MAX)	Variable-length text up to $2^{31} - 1$ bytes in length	<i>length</i> + 2 bytes
xml	Typed or untyped XML data	Depends on data stored

Creating Your Own Tables

Even though you never want to change, rename, delete, modify, annoy, pester, improve, or in any other way alter `aspnet_` tables, you can certainly *create* your own tables. Before you can put data into a table, of course, you have to create an empty table. To create your own SQL Server tables, work in Database Explorer — right-click on the Tables folder and choose Add New Table as shown in Figure 11-13.

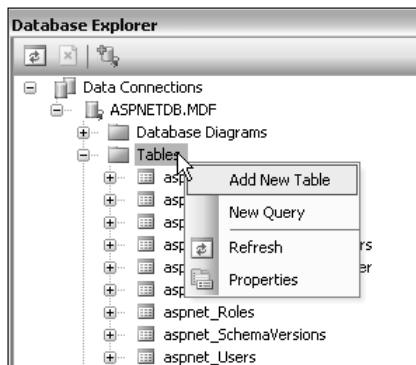


Figure 11-13:
Creating a
new table.

The next step is to define the columns (fields) that the table will contain; give each field a name of your own choosing. Make it a short name with no spaces or punctuation marks. Then you have to choose a data type for each field,

and decide whether or not the field can be left blank when adding new records to the table.

Defining a primary key

If the table you are creating will be on the “one” side of a one-to-many relationship, then that table must have a *primary key*. The primary key field must contain a value that’s unique to each record in the table. The field need not have the `uniqueidentifier` data type, but it should have some kind of field that’s automatically assigned a random, unique value each time a record is added to the table.



Another name for a primary key is *identity*, because the value in a record’s primary key field is also the record’s unique identity.

A good primary key should contain a value that’s assigned to the record randomly, so the key has no “meaning” to humans. Amateur database designers often make the mistake of trying to make the primary key a field that *is* meaningful to humans. Resist the temptation to do that; you’ll save yourself a world of headaches if you just let SQL Server manage the primary key for you, and keep it hidden from everyone.

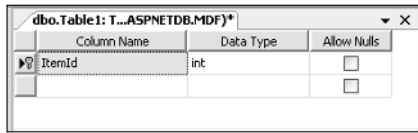
In the `Items` table shown way back in Figure 11-1, the `ItemId` field is the primary key. I didn’t use a GUID (although I could have). I used a five-digit number starting at 10001. That number is not something I typed in myself; it’s assigned to the record automatically each time a record is added to the table. Like a GUID, that number has no meaning — and, once entered, it can never be changed.

If you’re not going to use a GUID for a primary key, your next best bet would be an auto-numbered integer (`int`) field. That’s how I created the `ItemId` field in my sample `Items` table, and here are the steps required:

- 1. In the Column Name column, type a name for the field you’re about to create.**
- 2. In the Data Type column, choose one of the integer data types.**
If you’re not sure which to use, choose the `int` data type.
- 3. Clear the check mark (if any) from the Allow Nulls check box.**
A primary key cannot be left blank, so nulls cannot be allowed here.
- 4. Right-click the field name in the left column and choose Set Primary Key, or click the field name and choose Table Designer → Set Primary Key from the menu.**

You should see a small key symbol to the left of the field name. For example, in Figure 11-14, I created a field named `ItemId` that has the `int` data type, does not accept nulls, and is defined as the table’s primary key.

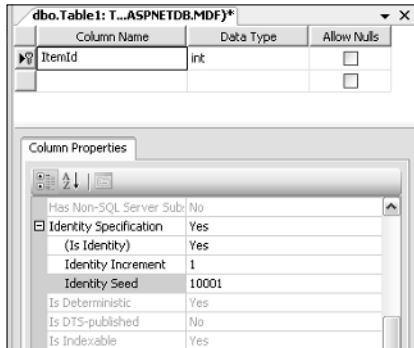
Figure 11-14:
Designating
a field as a
table's
primary key.



5. To have the records automatically numbered as they're entered into the table, first click the + sign next to **Identity Specification** in the **Column Properties** pane.
6. Set the **(Is Identity)** property to **Yes**.
7. Set the **Identity Increment** and **Identity Seed** properties to whatever number you want to appear first in the table.

In Figure 11-15, I've set the Identity Seed to 10001 and left the Identity Increment at 1. That means that the first record added to the table will automatically be assigned 10001. Subsequent records will be numbered 10002, 10003, 10004, and so forth.

Figure 11-15:
Here the
primary key
field will
be auto-
numbered,
starting at
10001.



Creating text fields

When you define a text field using any of the previously described `char` data types, you can use the `Length` property in the Column Properties to set the size of the field in characters. When you're using a fixed-length data type such as `char` or `nchar`, every field is stored using the exact number of characters you define as the field's `Length`. If you use one of the variable-length

data types, the `Length` property defines the maximum number of characters that can be stored in the field.

The `Items` table I'm building here could use at least three text fields, including one for an in-house `ItemId`. Although it would be a bad idea to use a "meaningful" field like that as a primary key, there's no harm in including it as a field in the table. For my example, I name the field `OurItemId` and make it a fixed-length text field (`char`) of 7 because my in-house codes all look like `BWG-101`, `BWG-102`, and so forth.

I'll also add a text field to store the name of the item, and another text field to store a lengthier description of the item. I'll name these fields `ItemName` and `ItemDescription`. There's no way to predict exactly how many characters an item name or description might be, so both of these will be variable-length text fields. I'll put a maximum length of 64 characters on the `ItemName` field, and a maximum length of 2,048 characters on the description. You should use field lengths that meet the needs of your own site's data storage.

To create text fields, follow these steps:

1. **Type the field name in the `Column Name` property.**
2. **Choose one of the text data types from the `Data Type` column's drop-down list.**
3. **Set the `Length` property in `Column Properties` to the maximum number of characters to be stored in the field (or just type a new length between the parentheses in the `Data Type` column).**

Figure 11-16 shows an example where I've created three more fields for the `Courses Items` table. You can see the `Length` property near the bottom of the figure in the `Column Properties` pane.

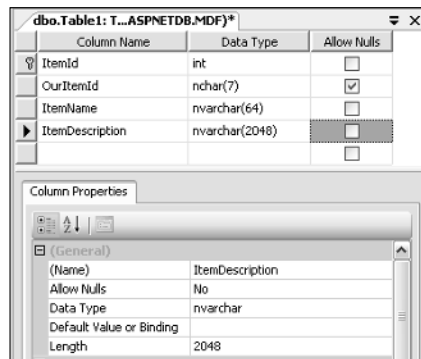


Figure 11-16:
Three text
fields added
to the table.

This is not an e-commerce book

If you're wondering when we're going to get to the part in this book where you actually sell things and do financial transactions, you can stop wondering. The answer is "never" — because there's really nothing built into Visual Web Developer to support that.

The more likely scenario is that you'd use some third-party service or tool to add e-commerce

capabilities. For example, you could create an account with Paypal and let people pay through that service. But that's just an example and not a topic that can be covered in this book. Sorry if my use of items, transactions, and money fields led you to believe otherwise.

Adding a money field

Data types for storing money don't require that you specify a `Length` property. It's a simple matter of entering a field name of your own choosing and choosing one of the money data types. For example, you could add a field named `ItemPrice` to the current table and give it the `smallmoney` data type (assuming you don't intend to charge \$200,000 or more for a course — if you're selling bigger-ticket items, use the `money` data type rather than `smallmoney`). So the finished sample table design would look like Figure 11-17.

Figure 11-17:
Here all
fields for the
Items table
are defined.

Column Name	Data Type	Allow Nulls
ItemId	int	<input type="checkbox"/>
OurItemId	nchar(7)	<input type="checkbox"/>
ItemName	nvarchar(64)	<input type="checkbox"/>
ItemDescription	nvarchar(2048)	<input type="checkbox"/>
ItemPrice	smallmoney	<input type="checkbox"/>
		<input type="checkbox"/>

Remember that my little `Items` table is just an example. Your own table could contain many more fields. But for the purposes of this example, we can consider that table finished.

Saving the new table

After you've defined the field names and data types for a table, click the Close (X) button in the upper-right corner of the Design surface. A dialog box asks if you want to save your changes. Choose Yes, and enter a name for your

table. (In my database, I named this table `Items`.) Then click OK. The new table will be listed along with other tables that are already in the database.

Creating the Transactions table

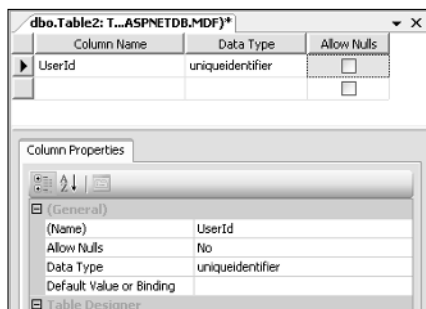
Creating the `Transactions` table is no different from creating the `Items` table. However, there are lots of rules and details that need to be dealt with. To get started, right-click the `Tables` folder and choose `Add New Table`. You get the standard design screen for defining field names and data types.

How you define the first two fields in the transaction table is critical. Because if you make even the slightest mistake, it won't work. In the `Transactions` tables, the two key fields are actually foreign keys, not primary keys. (A *foreign key* has the same name and data type as a primary key in a table, on the "one" side of a one-to-many relationship.)

For example, there is a one-to-many relationship between users and transactions. In the `Users` table, the `UserId` field is the primary key that uniquely identifies each user. In the `Transactions` table, the `UserId` field (alone) cannot be marked as a primary key, and should never be automatically assigned a value using `newid()` or an `Identity Specification`.

So, the first field in this example's `Transactions` table must be named `UserId`, must be the `uniqueidentifier` data type, and must have its `Allow Nulls` check box blank (because records with no value in this field would be useless). The `Default Value or Binding` property for the field must be left empty as shown in Figure 11-18.

Figure 11-18:
First field
in the
Transac-
tions
table
defined.

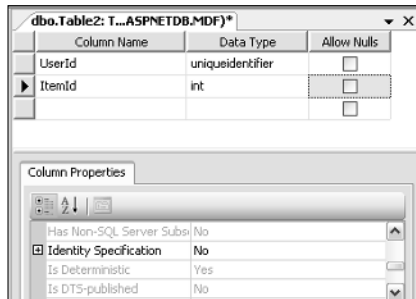


The `Transactions` table also needs a field that provides a link to the primary key in the `Items` table. The primary key in the `Items` table is named `ItemId` and its data type is `int`. In the `Transactions` table, the `ItemId` field

must not be auto-numbered and must not allow nulls. So its `Identity Specification` property must be set to `No`, as shown in Figure 11-19.

At this point in time, the `Transactions` table has all the fields it needs to record transactions and provide the links between users and items. Even though the table must contain the two fields just created here, there's no rule that says it can't contain other fields as well.

Figure 11-19:
Second field
in the
Transac-
tions
table.



For example, if you wanted to record the date of each transaction, add a `smalldatetime` field. If you want to keep track of whether a transaction has been paid, add a `bit` field. If items are such that a user might buy several of them at a time (say, pencils or kumquats), your `Transactions` table could include a `Qty` field to indicate how many items were purchased. If prices of items change over time, and you want to know exactly what the user paid in each transaction, you could include a `SellingPrice` field that records exactly what the user paid for the item.

For the example in this book, I added two fields to my `Transactions` table: `TransactionDate` and its data type `smalldatetime`. The idea is to record the date and time of each transaction. I also added an `IsPaid` field and gave it the `bit` data type. In records, that field contains `True` for transactions that have been paid, and `False` for unpaid transactions. Figure 11-20 shows that sample `Transactions` table with the two additional fields defined.

Figure 11-20:
This sample
Transac-
tions table
contains
four fields.

Column Name	Data Type	Allow Nulls
UserId	uniqueidentifier	<input type="checkbox"/>
ItemId	int	<input type="checkbox"/>
TransactionDate	smalldatetime	<input type="checkbox"/>
IsPaid	bit	<input type="checkbox"/>

A Primary Key for Transactions

As mentioned, if a table is on the “one” side of a one-to-many relationship, then that table must have a primary key to uniquely identify each record. In that regard, the `Transactions` table does not need a primary key, because it’s on the “many” side of its relationships with users and items.

However, there is another rule in Visual Web Developer that states that if you want to be able to edit a table through a Web page, then that table must have a primary key. So if you want to be able to edit the `Transactions` table through Web pages, you’ll need to give that table a primary key.

As always, the primary key need not have any special “meaning” — it can just be a number that’s automatically assigned to each new record, sort of like an order number on a paper order form. An automatically incremented `int` (integer) field will do nicely.

Figure 11-21 shows an example where I’ve added a field named `TransactionId` to the `Transactions` table. As you can see in its Column Properties, I set the fields `Is Identity` to `Yes`, set the `Identity Increment` to `1`, and the `Identity Seed` to `20000`. You can use any starting number you like. I just opted to number transactions `20000`, `20001`, `20002`, and so forth arbitrarily. I also right-clicked the field name and chose `Set Primary Key` to make the field the table’s primary key.

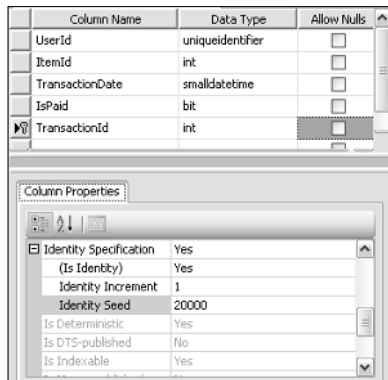
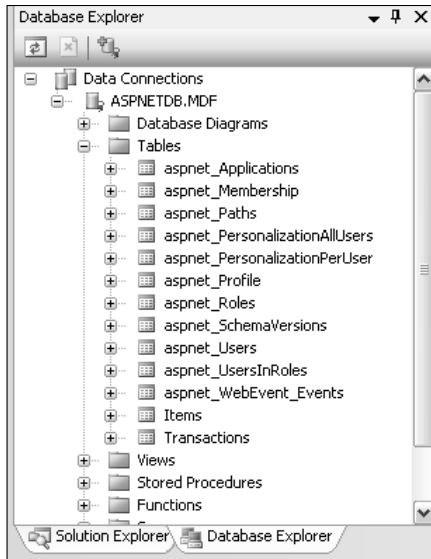


Figure 11-21:
Final structure of the `Transactions` table.

When you’re done defining fields for the `Transactions` table, close and save the design as you would any other object. In my example, I named the table `Transactions` — as shown in Figure 11-21 — and it will be added to the list of other tables already in the database. For example, Figure 11-22 shows my list of tables after creating the new tables named `Items` and `Transactions`.

Figure 11-22:
Two new
tables
(**Items** and
Transactions),
listed with
other tables.



Populating Tables

When you first create a table, it will be empty. So when you right-click the table name and choose Show Table Data, you'll see an empty record with the word `NULL` in each field. For example, right-clicking an empty `Items` table and choosing Show Table Data displays that table's contents. But because the table is brand new and empty, its contents consist of a single empty record that has the word `NULL` (which means, in effect, *blank*) in each field, as shown in Figure 11-23.

Figure 11-23:
Here's the
Empty
Items
table, open
and really
empty.

ItemId	OurItemId	ItemName	ItemDescription	ItemPrice
NULL	NULL	NULL	NULL	NULL

You can add data to the table just by replacing the `NULL`s with the data you want to store. That's not the only way to do it, or even the best way. For example, you'd never want to type the data manually to populate the `aspnet_` tables in the database (trust me on that one). You must use the Web Site Administration Tool, `CreateUserWizard`, and so forth to manage that data.

But, if you need some sample data to play around with in the tables you created yourself, you can manually type that data directly into the tables. It's not the most intuitive way to do it; there are some rules you have to play by.



Never use the procedure described here to add data to any table whose name starts with `aspNet_`. Always use the Web Site Administration Tool to add, edit, and delete data in those tables. Use the method described only to enter some sample data into a table you created yourself.



Here are some quick tips for entering data into fields:

- ✓ When you enter data into a record that contains an automatically-generated primary key (for example, the `ItemId` field in the `Items` table), you must leave that field empty (`NULL`). The field gets filled with a value automatically, after you've filled in other fields in the record.
- ✓ When you're typing data into table fields, *do not* press Enter when you complete your entry. Instead, when you've finished filling one field and want to move on to the next, press the Tab key. Or just click the next field in which you want to enter or edit data.
- ✓ The little pencil that appears after you fill in a field isn't part of the data in the field. It's only there to tell you the field has been edited since the table was first opened.
- ✓ When typing dollar amounts, don't type the dollar sign. There's no need to bother typing commas, either; they'll just be removed. For example, if you want to enter \$1,234.56 into a money field, type `1234.56`. In the Design surface, all dollar amounts have four digits to the right of the decimal point and no dollar sign. Don't let that bother you. As long as you're working in Database Explorer, how the data *looks* isn't important.
- ✓ If you don't have data to fill a field yet, you can leave the field empty (`NULL`), provided you allowed nulls in the design of the table. If you didn't, and you find that the table really needs to allow nulls in a field, go back to the table's definition and select the Allow Nulls check box for that field.
- ✓ After you've filled in all the fields for a record, you can press Tab or Enter to complete that record. SQL Server validates the data and saves the record. If there are any problems, the record won't be saved. Instead, you'll see an error message (stating `No row was updated`) with a description of the problem. You have to click OK in that message box and fix the problem. This fix often involves clicking the faulty value and then pressing Escape to "undo" the entry currently in the column.

The example in Figure 11-24 shows where I've typed some sample (and fake) data into the `Items` table. The `ItemDescription` field contains much more text than can be displayed in the narrow column. But that text isn't important here.

Figure 11-24:
Adding
some
records to
my **Items**
table.

Items: Query(...\ASPNETDB.MDF)					
	ItemId	OurItemId	ItemName	ItemDescription	ItemPrice
	10001	BWG-101	Introduction to Bird Watching	Stop being a PC ...	29.9500
	10002	BWG-201	Intermediate Bird Watching	Once you've ma...	29.9500
	10003	BWG-301	Advanced Bird Watching	Renowned ornit...	39.9500
	10004	RWG-101	Introduction to Reptile Watching	Sick of birds? Ho...	29.9500
	10005	RWG-401	Recovering from Reptile Wounds	Reptiles can stin...	49.9500
	NULL	NULL	NULL	NULL	NULL



To widen or narrow a column, click and hold on the bar to the right of that column's name, and then drag left or right.

I'll need some sample data in my **Transactions** table to illustrate upcoming concepts. So I opened that table and added the records below. It's imperative that the value in each record's **UserId** field exactly matches an actual user's GUID in the **aspnet_Users** table. So, to enter GUIDs into records, I copied and pasted actual GUIDs from the **aspnet_Users** table. The **TransactionId** values are entered automatically. So when you get to that field, just press the **Tab** key to move to the next record without typing a **TransactionId** number.



Dummies beware! Typing the GUIDs shown in Figure 11-25 into your own table will not work. Every GUID you manually enter into your own **Transactions** table must exactly match an actual GUID from your own **aspnet_Users** table.

Figure 11-25:
Sample data
in the
Transac-
tions
table.

	UserId	ItemId	TransactionDate	IsPaid	TransactionId
	f572a1ed-3553-40c7-99ce-1ad3bcece831	10002	6/15/2006 12:00:00 AM	True	20000
	f572a1ed-3553-40c7-99ce-1ad3bcece831	10003	6/21/2006 12:00:00 AM	True	20001
	f572a1ed-3553-40c7-99ce-1ad3bcece831	10005	6/25/2006 12:00:00 AM	True	20002
	8a116cb2-4503-4f83-870e-4fa50bee946a	10003	6/3/2006 12:00:00 AM	True	20003
	00a35f8e-34ce-4fb1-b7b4-8f2792e9b799	10002	6/17/2006 12:00:00 AM	True	20004
	00a35f8e-34ce-4fb1-b7b4-8f2792e9b799	10003	6/21/2006 12:00:00 AM	True	20005
	66cd551e-470c-4e57-9de3-d197bb6abc98	10003	6/1/2006 12:00:00 AM	True	20006
	66cd551e-470c-4e57-9de3-d197bb6abc98	10004	6/2/2006 12:00:00 AM	True	20007
	66cd551e-470c-4e57-9de3-d197bb6abc98	10005	6/20/2006 12:00:00 AM	True	20008
	bf93fd26-d11c-446c-b065-fd1de82182dc	10003	7/1/2006 12:00:00 AM	True	20009
	bf93fd26-d11c-446c-b065-fd1de82182dc	10004	7/21/2006 12:00:00 AM	True	20010
	bf93fd26-d11c-446c-b065-fd1de82182dc	10005	7/27/2006 12:00:00 AM	True	20011
	NULL	NULL	NULL	NULL	NULL

At this point, I have all the tables that the membership system created, plus my **Items** and **Transactions** tables. There is a natural many-to-many link between users in the **Membership** table and items in my **Items** table. The **Transactions** table contains the fields necessary to link many users to many items, and also provides a record of who purchased what and when.

At this moment, SQL Server is not “aware” that there are relationships among these tables. You have to describe the relationships by linking tables together in queries.

Linking Tables

Although the tables in a database store data, the way you get exactly the data you need, and only the data you need, when you need it, is through Structured Query Language, abbreviated SQL and often pronounced like *sequel*. You use SQL to create *queries* (also called SQL statements or `SELECT` statements) — commands that describe exactly what data you want to retrieve from the database, and where to find that data. Here is an example of a simple SQL statement:

```
SELECT * FROM Items
```

This SQL statement, when executed, retrieves all columns and all records from the table named `Items`. (The `*` is short for “all columns.”)

If you want to retrieve only certain columns of data from a table, replace the `*` with the names of the columns the SQL statement should retrieve. For example, the following SQL statement retrieves data from the `UserId` and `UserName` (only) columns in the `aspnet_Users` table:

```
SELECT UserId, UserName FROM aspnet_Users
```

Both sample SQL statements just given retrieve all records from the table. The only time such an extraction doesn’t happen is when that SQL statement includes a `WHERE` clause, which specifies which records to retrieve. The `WHERE` clause is often called a *filter* because it “filters out” any records you don’t need. (Or, more accurately, it filters out the records that the current Web page doesn’t need to show.)

A `WHERE` clause limits the records retrieved from the data to those records that meet some criterion. For example, this SQL statement retrieves exactly one record (not all records) from the `Items` table: The record that has 10002 in its `ItemId` field:

```
SELECT * FROM Items WHERE ItemId = 10002
```

If you wanted just the `ItemName` and `ItemDescription` fields from the table, and only the records for item 10002, then you replace the `*` with the names of columns you want, as follows:

```
SELECT ItemName, ItemDescription FROM ItemsTable WHERE ItemId = 10002
```

Not to worry: You don’t actually have to write the SQL statements yourself. And that’s a good thing — SQL statements can be far more complex than the examples shown here. To avoid all that tedious and error-prone typing, you can use the Query Builder to create your complex SQL statements. The Query Builder lets you pick and choose what you want to extract from your tables with ease — and with a more intuitive graphical user interface. You

pick and choose options, the Query Builder then writes the appropriate SQL statement for you.

The Query Builder appears automatically whenever you perform some action that requires getting data from a database. You'll see examples in upcoming chapters; in this chapter, as a general example, we use the Query Builder to link tables together into a *view*.

Creating a view

In SQL Server, a view is a saved query. When you set up your site for Membership, VWD automatically created several views. They are listed under Views in Database Explorer. The name of each automatically-created view starts with `vw_aspnet_` as shown in Figure 11-26.

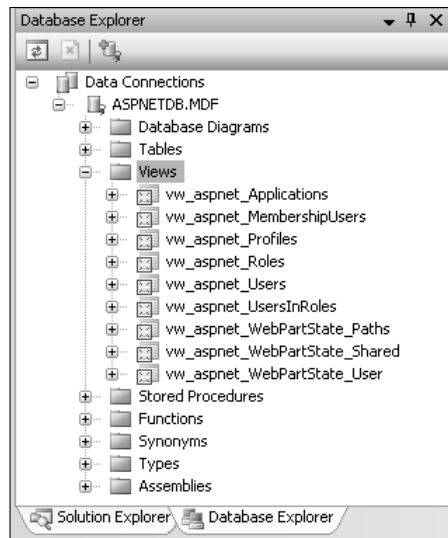


Figure 11-26:
Views
created
by the
membership
system.



Never delete, change, or rename any view whose name starts with `vw_aspnet_`. Those views are created by the membership system, *for* the membership system, and they really don't like being monkeyed around with.

To illustrate using the Query Builder to link users, transactions, and items, I'll create a view named `UsersAndItemsView`. You create a view as you would

any other object: Right-click the Views folder in Database Explorer and choose Add New View. A dialog box named Add Table opens.

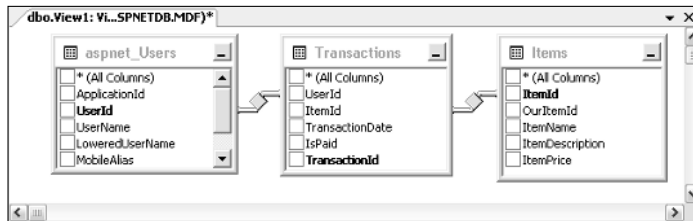
The first step in building the query is to choose the tables from which the query will get data. If there's a many-to-many relationship among selected tables, the view must also include the `Transactions` table that provides the link between tables. To add a table to the query, click its name in the Add Table dialog box, and then click OK. (For this example, I add the `aspnet_Users`, `Transactions`, and `Items` tables to the view.) Click the Close button in the Add Tables dialog box after choosing your tables.



If you forget to add a table before clicking the Close button, choose Query Designer → Add Table from the menu bar, or click the Add Table button in the toolbar.

Each table you add appears as a field list in the Diagram pane at the top of the Query Builder. Each field list shows the names of fields within the table. You can move and size the field list's little windows by using standard techniques (drag the title bar to move, drag any corner or edge to size). In most cases, the lines connecting tables by their key fields are added to the field lists, as in Figure 11-27.

Figure 11-27:
Three tables
joined in
the Query
Builder.



With the three items linked together, the next step is to choose which fields you want to see from these tables. You choose which fields you want the view to display by clicking the check box next to each desired field name. As you do so, each field name you check appears in the Criterion pane, just below the Diagram pane.

Below the Criterion pane is the Show SQL pane, which shows the actual SQL statement that the Query Builder creates as you go. You can size those panes by dragging their borders up and down. You can choose which panes you want to hide by using buttons in the toolbar shown near the top of Figure 11-28; the figure also shows how the query looks after you choose some field names from tables in the Diagram view.

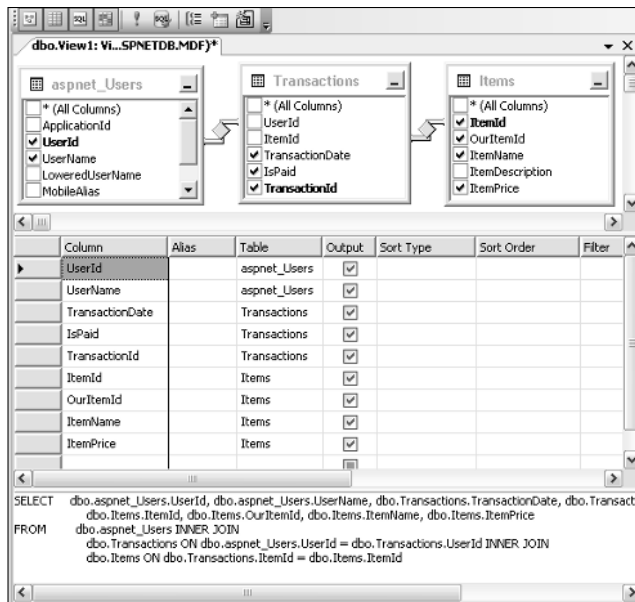


Figure 11-28:
Fields that the query should retrieve are selected.

While you're designing a query, you can test it out at any time using any of these methods:

- Right-click some empty space in the Design surface and choose **Execute SQL**.
- Press **Ctrl+R**.
- Click the **Execute Query** button (red exclamation mark) in the toolbar.
- Choose **Query Designer** → **Execute Query** from the menu bar.

The results of executing the query appear in the Results pane at the bottom of everything else. Figure 11-29 shows the results of executing the query from Figure 11-28. Although it might not look like much, at first glance, it could be the solution to many of your data processing problems.

UserId	UserName	TransactionDate	IsPaid	TransactionId	ItemId	OurItemId	ItemName	ItemPrice
F572a...	Bob	6/15/2006 ...	True	20000	10002	BWG-201	Intermediate Bird Watching	29.9500
F572a...	Bob	6/21/2006 ...	True	20001	10003	BWG-301	Advanced Bird Watching	39.9500
F572a...	Bob	6/25/2006 ...	True	20002	10005	RWG-401	Recovering from Ruptile Wounds	49.9500
8a215...	Neville	6/3/2006 L...	True	20003	10003	BWG-301	Advanced Bird Watching	39.9500
00a35...	Elsie	6/12/2006 ...	True	20004	10002	BWG-201	Intermediate Bird Watching	29.9500
00a35...	Elsie	6/21/2006 ...	True	20005	10003	BWG-301	Advanced Bird Watching	39.9500
66cd5...	Carol	6/1/2006 L...	True	20006	10003	BWG-301	Advanced Bird Watching	39.9500
66cd5...	Carol	6/2/2006 L...	True	20007	10004	RWG-101	Introduction to Reptile Watching	29.9500
66cd5...	Carol	6/20/2006 ...	True	20008	10005	RWG-401	Recovering from Ruptile Wounds	49.9500
bf93f...	Alice	7/1/2006 L...	True	20009	10003	BWG-301	Advanced Bird Watching	39.9500
bf93f...	Alice	7/21/2006 ...	True	20010	10004	RWG-101	Introduction to Reptile Watching	29.9500
bf93f...	Alice	7/27/2006 ...	True	20011	10005	RWG-401	Recovering from Ruptile Wounds	49.9500

Figure 11-29:
Results of executing the query.

Each record in the view's results represents a single transaction; the view always shows one record for every transaction made, up to the moment you see the results. In other words, if you ignored all fields except `UserId` and `ItemId`, you'd see that the view contains exactly the same records as the `Transactions` table.

However, the view offers a big advantage over the `Transactions` table alone. Each record in the view includes the user name, date of the transaction, paid status, `OurItemId`, `ItemName`, and `ItemPrice` fields. In other words, the view contains a lot more useful information that the `Transactions` table provides — and offers greater control of your data in these ways:

- ✓ The way the data *looks* in the query results is of no importance. Users never see query results like those you're seeing in Web pages. They see the information in whatever format you present it to them — and you have near-infinite choices there.
- ✓ Any time you “call upon” a view for information, it returns all the fields shown in the current query results — including current data from the tables — but the view doesn't “contain” any data at all. Each time it's executed, it has no choice but to get the specified information from the tables that are currently in the database.
- ✓ A view can only be used to retrieve data from a table — not to edit data in the underlying table. That's an advantage because in a Web site, nine times out of ten you want a Web page to *show* data to the user but not to allow the user to *edit* the data. Why? Well (for instance), imagine an unscrupulous user editing the page to change the price of an item in your `Items` table to a penny — and then ordering 100 of them. . . .

Using a view to access data from the tables provides the added security of knowing that a user cannot gain any access whatsoever to data in the tables from which the view retrieves its records. Also, the fact that the Web server doesn't need to fuss with editing improves overall performance; data moves faster to the user's page from the server.

To save a view, click the Close button in its upper-right corner and choose Yes when asked about saving your changes. Name the view (I'll call this one `UsersAndItemsView1`) and click OK. The view name is added to the list of views in Database Explorer.

A more detailed view

One weakness of the previous view is that it didn't include the user's e-mail address. It can't; the linked `aspnet_Users` table doesn't contain the user's e-mail address. That bit of information is stored in the `aspnet_Membership` table.

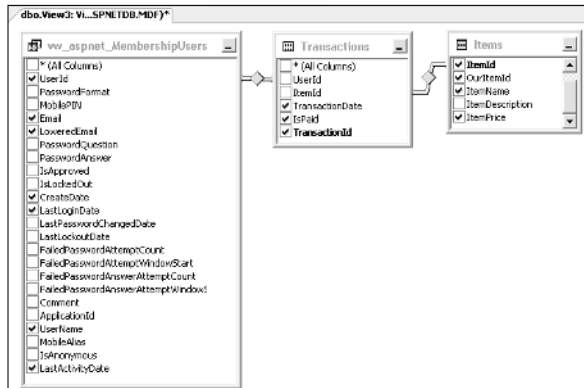
There is, however, a view named `vw_aspnet_MembershipUsers` already defined under Views in Database Explorer. You can use that view in place of the `aspnet_Users` table to create a query that includes users' e-mail addresses.

Creating another view named `DetailedUsersItemsView` illustrates the point:

1. Right-click the Views folder and choose **Add New View**.
2. In the Add Tables dialog box, click the Views tab, click `vw_aspnet_MembershipUsers`, then click the **Add** button.
3. In the Add Tables dialog box, click the Tables tab.
4. Click the `Transactions` table, then click **Add**.
5. Click the `Items` table, then click **Add**.
6. Click **Close** in the Add Tables dialog box.
7. In the diagram pane, drag the `UserId` field in the `vw_aspnet_MembershipUsers` field list to the `UserId` field name in the `Transactions` table. The two fields will then be connected by a line as shown in Figure 11-30.

In Figure 11-30, I sized and arranged things so you can see all the field lists and the lines joining the tables by their key fields.

Figure 11-30:
The
Detailed
Users
Items
View's
tables and
selected
fields.



The query can return any fields from any table you see. In Figure 11-30, I chose some fields that might come in handy in the future. However, you can choose any fields you like.

Executing the query returns a set of records similar to those from the less-detailed query — exactly one record per transaction shows up in the query

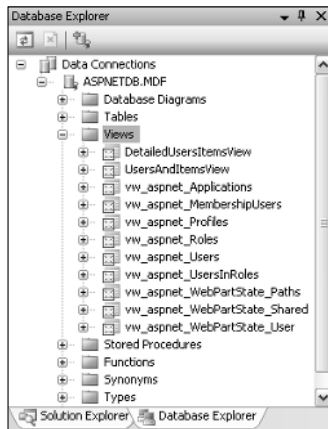
results. However, this view offers more information about who made each purchase — including each user’s e-mail address, the date he or she created the account, the date of the user’s last login, and more. Figure 11-31 shows the results — but actually the results table is wide enough to see all the columns. Suffice it to say that for every checked field name in Figure 11-30, there’s a field containing data in the query results.

Figure 11-31:
Results of
executing
the detailed
query.

UserId	Email	Comp...	CreateDate	LastLogin...	UserName	LastActivDate	TransactionId	ItemId	OutItemId	ItemName	ItemPrice	TransactionDate	IsPaid
291e...	bob@evoed.com	bob...	7/9/2005 ...	7/14/200...	Bob	7/15/2005 8:01...	20000	10002	BWG-301	Intermediate Br...	29,9500	6/15/2006 12:...	True
f572...	bob@evoed.com	bob...	7/9/2005 ...	7/14/200...	Bob	7/15/2005 8:01...	20001	10003	BWG-301	Advanced Bird ...	39,9500	6/21/2006 12:...	True
f572...	bob@evoed.com	bob...	7/9/2005 ...	7/14/200...	Bob	7/15/2005 8:01...	20002	10005	RWG-401	Recovering from...	49,9500	6/25/2006 12:...	True
0a1...	erns@coaher...	erns...	6/6/2005 ...	7/11/200...	Newbie	7/15/2005 7:09...	20003	10003	BWG-301	Advanced Bird ...	39,9500	6/17/2006 12:...	True
00a...	eks@eveEd.com	eks@...	7/9/2005 ...	7/11/200...	Eaks	7/15/2005 7:03...	20004	10002	BWG-201	Intermediate Br...	29,9500	6/17/2006 12:...	True
00a...	eks@eveEd.com	eks@...	7/9/2005 ...	7/11/200...	Eaks	7/15/2005 7:03...	20005	10003	BWG-301	Advanced Bird ...	39,9500	6/21/2006 12:...	True
66c...	carol@evoed.com	carol...	7/9/2005 ...	7/11/200...	Carol	7/15/2005 7:20...	20006	10003	BWG-301	Advanced Bird ...	39,9500	6/17/2006 12:...	True
66c...	carol@evoed.com	carol...	7/9/2005 ...	7/11/200...	Carol	7/15/2005 7:20...	20007	10004	RWG-101	Introduction to ...	29,9500	6/17/2006 12:...	True
66c...	carol@evoed.com	carol...	7/9/2005 ...	7/11/200...	Carol	7/15/2005 7:20...	20008	10005	RWG-401	Recovering from...	49,9500	6/20/2006 12:...	True
bf93...	alice@evoed.com	alice...	7/9/2005 ...	7/11/200...	Alice	7/11/2005 12:0...	20009	10003	BWG-301	Advanced Bird ...	39,9500	7/11/2006 12:0...	True
bf93...	alice@evoed.com	alice...	7/9/2005 ...	7/11/200...	Alice	7/11/2005 12:0...	20010	10004	RWG-101	Introduction to ...	29,9500	7/21/2006 12:...	True
bf93...	alice@evoed.com	alice...	7/9/2005 ...	7/11/200...	Alice	7/11/2005 12:0...	20011	10005	RWG-401	Recovering from...	49,9500	7/27/2006 12:...	True
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

For purposes of future examples, I saved the query shown in Figure 11-30 as `DetailedUsersItemsView`, so in my own Database Explorer, that name has been added to the list of Views in the current database (as in Figure 11-32). To execute a view to see only the data it retrieves — without seeing its design — you can right-click the view name and choose `Show Results`.

Figure 11-32:
Two new
Views
added to the
database.



The way the data looks in the view results is of zero importance. Formatting takes place when you display data from a view in a Web page.

In real life, there would rarely be any need to open a view from Database Explorer and see the records it returns, because the way the results are shown in Database Explorer aren't especially pretty or easy to read. Instead — normally — you'd use the view to extract data from your tables via data-bound controls in the Web pages you create.

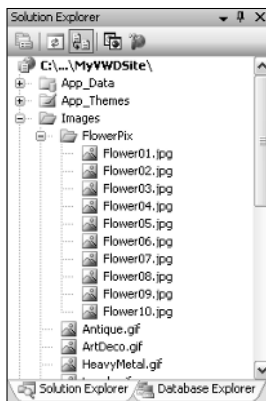
Each data-bound control can choose as much (or as little) information from the view as it needs. For example, one control could display a list of all users who purchased a specific item. Another could list all items purchased by a specific user. Still other controls could organize information from the view in other ways.

Creating a Table of Pictures

Not all database tables need to be linked to other tables. You can use a database table to store things that don't relate directly to specific users or specific items. For example, if you have pictures you want to show or allow people to download, you can create a table of pictures. Then, as you'll see in Chapter 12, you can use Data controls to make it easy to display multiple pictures on a page.

You don't need to put the pictures directly in the database. Rather, you can store all the pictures in a folder (or multiple folders). Then just create a table that identifies the picture's location within your site. As an example, Figure 11-33 shows where I've created a subfolder named FlowerPix within my Images folder in Solution explorer. Each of the filenames, Flower02.jpg, Flower02.jpg, and so forth, is a photo of a flower.

Figure 11-33:
The FlowerPix folder contains ten pictures of flowers.



Next, you can create a table that contains three fields. One field will be a primary key that just assigns each record a unique number. Even though this table won't be on the "one" side of a one-to-many relationship, a table needs to have a primary key if you want to be able to edit it through Web pages. So we'll add a primary key to the table we're about to create.

The second field will be a caption or title for each photo. The third field will be the path and filename of a photo. The path to each picture will look something like this:

```
~/Images/FlowerPix/Flowerxx.jpg
```

where `~/Images/FlowerPix` means "the FlowerPix folder in the Images folder, under the root (`~`) folder." The `Flowerxx.jpg` will be the name of a specific picture within that folder. I'll name the sample table `Photos`. You create it as you would any other table. Here are the steps:

1. If you're in Solution Explorer, click the Database Explorer tab.

2. Right-click on Tables under the database name and choose Add New Table.

3. Type `PhotoId` as the first field name, and make it the `smallint` data type.

If you think you'll have more than 32,000 pictures, use the `int` data type rather than `smallint`.

4. Clear the check mark from the `Allow Nulls` field.

5. In the Column Properties, click the + sign next to Identity Specification.

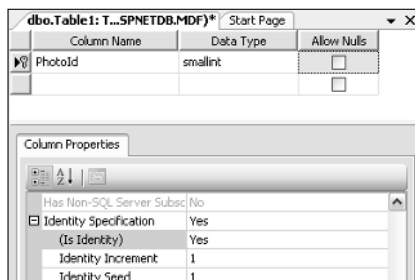
6. Set the `(Is Identity)` property to `Yes`.

You can leave the Identity Increment and Identity Seed each set to 1, so records are just numbered 1, 2, 3, 4, and so forth.

7. Right-click the `PhotoId` field name and choose Set Primary key.

Figure 11-34 shows how the table design should look so far.

Figure 11-34:
The `PhotoId` field is an auto-numbered primary key.



8. Add a second field named `PhotoCaption`, and set its Data Type to `varchar(50)`.
9. Add a third field named `PhotoURL` and set its Data Type to `varchar(64)`.

Figure 11-35 shows the final structure of the table.

Figure 11-35:
Completed
Photos
table
design.

Column Name	Data Type	Allow Nulls
PhotoID	smallint	<input type="checkbox"/>
PhotoCaption	varchar(50)	<input type="checkbox"/>
PhotoURL	varchar(64)	<input type="checkbox"/>



When defining a field that stores any kind of URL or address, use either `varchar` or `nvarchar`. Using `char` or `nchar` will pad short addresses with blank spaces, which in turn may prevent the link from working properly later when used on a page.

10. Click the **Close (X)** button in the upper-right corner of the Design surface.
11. Choose **Yes** when asked about saving your changes.
12. Change the name of the table to `Photos`, then click **OK**.

Now you have a new table, named `Photos`, in the database. Each record in the table can contain a photo caption and the link to a photo. If you already have your pictures in a folder, you can right-click the table and choose **Show Table Data**.



Because `PhotoID` is an automatically-numbered field, you want to leave it as **Null** when entering new records. Just type a caption name and a link to a picture in the `PhotoCaption` and `PhotoURL` fields. Then type a caption and link for each photo.

For my example, where each photo is a picture of a flower, I put a (fake) Latin name for each photo into the `PhotoCaption` field, and a link to each photo in the `PhotoURL` field. (I don't know the actual Latin names for the flowers, so I just made some up, borrowing a few names from animal anatomy.) Figure 11-36 shows an example. I've opened Solution Explorer in that figure so you can see how the `PhotoURL` in each record refers to a specific image in the `~/Images/FlowerPix` folder.

Figure 11-36:
Some sample records in my `Photos` table.

PhotoId	PhotoCaption	PhotoURL
1	Wacha Macallit	~/Images/FlowerPic/Flower01.jpg
2	Medulla Oblongata	~/Images/FlowerPic/Flower02.jpg
3	Pinkus Daisius	~/Images/FlowerPic/Flower03.jpg
4	Latissimus Dorsi	~/Images/FlowerPic/Flower04.jpg
5	Blancus Cupis	~/Images/FlowerPic/Flower05.jpg
6	Floris Wateverish	~/Images/FlowerPic/Flower06.jpg
7	Rosas Notaclue	~/Images/FlowerPic/Flower07.jpg
8	Corpus Callosum	~/Images/FlowerPic/Flower08.jpg
9	Ipssem Lorem	~/Images/FlowerPic/Flower09.jpg
10	Dingus Battius	~/Images/FlowerPic/Flower10.jpg
*	NULL	NULL

The table doesn't look like much with just the text in it. However, keep in mind that a table is always just "raw data" and how things look in the table doesn't dictate how they'll look on a page. As you'll discover in Chapter 12, you can use a `DataList` control with the `Photos` table to show the actual pictures and captions on a page. For now it's sufficient to just close and save the table.

Creating a Table of HyperLinks

If your site has many links to Web sites outside your own site, you can store all those links in a database table. Doing so keeps all the links in one place, where they're easy to manage. You can then use whatever links you want on any page you want without having to worry about keeping links up-to-date across many different pages within your site.

For a table of links, you'll need at least two text fields, one for the name or title of the site, and one for the site's URL (address). If you like, you could add a third field for storing a description of each site. Also, if you want to be able to edit the table's data through Web pages, the table will need a primary key.

To keep things fairly simple, I'll show you how to create a table that contains a primary key, a field for storing each site's name, and a field for storing each site's address. The steps are:

1. If you're in **Solution Explorer**, click the **Database Explorer** tab.
2. **Right-click on Tables** under the database name and choose **Add New Table**.
3. **Type SiteId as the first field name, and set its Data Type to `smallint`.**

If you think you'll have more than 32,000 links, use the `int` data type rather than `smallint`.

4. Clear the check box for Allow Nulls.
5. In the Column Properties pane, click + next to Identity Specification.
6. Set the (Is Identity) property to Yes.

You can leave the Identity Increment and Identity Seed fields each set to one.

7. Right-click the `siteId` field name and choose Set Primary Key.

Figure 11-37 shows the table structure so far, with just the auto-numbered primary key `siteId` defined.

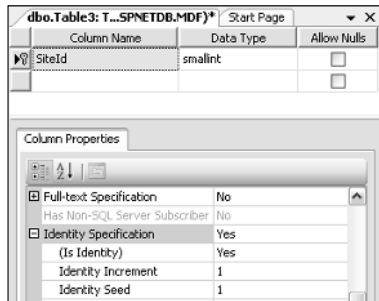


Figure 11-37:
The primary key for a table of hyperlinks.

8. Add a second field to store the site's title. For my example, I'll name that field `siteName` and set its Data Type to `nvarchar(50)`.
9. Add a third field for storing the site's URL. In my example, I'll name it `siteURL` and give it the `varchar` Data Type with a maximum length of 64 characters.

Figure 11-38 shows my completed table structure.

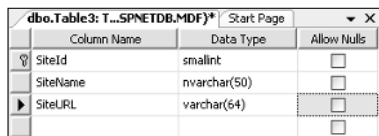


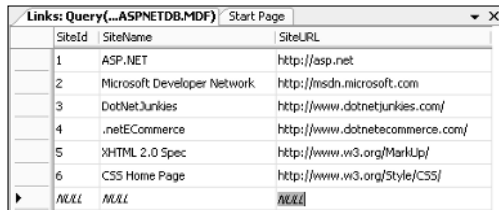
Figure 11-38:
Structure of my sample Links table.

10. Click Close in the upper-right corner of the Design surface.
11. When asked about saving the table, choose Yes.
12. Name the table `Links` and click OK.

To add data to the table, right-click its name in Database Explorer and choose Show Table Data. When typing in data, remember to leave the `SiteId` field set to Null, as it will be numbered automatically after you've filled the other fields. You can type any text for the site's name or title. But when specifying the site's URL, make sure you use the full `http://...` address.

Figure 11-39 shows an example where I've put in some site titles and the links to those sites. The links don't actually do anything in the table. Remember, the table is just a means of storing the data. But as you'll discover in Chapter 12, you can use a `DataList` control to create a page that displays each site title as a link that takes the user directly to the site.

Figure 11-39:
Storing links
in a table.



SiteId	SiteName	SiteURL
1	ASP.NET	http://asp.net
2	Microsoft Developer Network	http://msdn.microsoft.com
3	DotNetJunkies	http://www.dotnetjunkies.com/
4	.netECommerce	http://www.dotnetecommerce.com/
5	XHTML 2.0 Spec	http://www.w3.org/MarkUp/
6	CSS Home Page	http://www.w3.org/Style/CSS/
NULL	NULL	NULL

Thus ends the world's quickest SQL Server/Database Design crash course. There is a great deal more to know about those topics. (But you knew that.) What you've discovered here should be enough to get you started. If nothing else, it has covered a lot of things many other VWD information resources will assume you already know. Chapter 12 shows many ways of putting the data to good use within the site.

Chapter 12

Using Data in Web Pages

In This Chapter

- ▶ Binding data to pages
 - ▶ Using the Data Source Configuration Wizard
 - ▶ Using the `GridView` and `DetailsView` controls
 - ▶ Using the `DataList` and `FormView` controls
-

Visual Web Developer is a tool for building dynamic, data-driven Web sites — *dynamic* because the information you display on these Web pages doesn't have to be the same for everyone. You can get your site to create a page that's appropriate for every user, showing only the appropriate (user-specific) data from a database.

Take a big search engine like Google for instance. When you submit a search string to Google, you're actually submitting a query to its database. What its database spits back are pages and pages of links to Web pages that contain the word or phrase you searched for. In other words, Google's Web site *dynamically* created the page you see using data from its database.

To display data from your site's database in Web pages, you *bind* data from the database to controls on the page. While sitting on your server, the control is just a placeholder that contains no data. When a Web site visitor requests the page, the control can then be filled with whatever data is appropriate to that specific request.

In other words, if 100 different people visit the page, each might see something different, just as 100 different people doing different Google searches see 100 different lists of things in the page that Google sends them. What makes the page *dynamic* is that the page contains only data that is relevant to the user's request — and that changes from user to user.

This chapter explores how to use dynamic content in your own Web pages: how to bind the data to controls and how to place that data into your Web site.

Binding Data to Controls

You can bind data to all kinds of controls in Visual Web Developer. But before I get into specific controls, it's worth reviewing some things that apply to all data-bound controls. For openers, you always put data-bound controls on Web form (.aspx) pages. So the first step is to open or create, in Design view, the page on which you want to place the control.

Next, you drag a data-bound control from the Toolbox onto the page (or, if the current page has a Master Page, you drag the control into the Content area of the page). The control won't look like much in Design view. It's just a placeholder; don't be alarmed by its ugly appearance.

The control won't automatically be bound to anything specific in your database. You have to tell it what to bind to. For this you can use the Data Configuration Wizard

Using the Data Configuration Wizard

To bind a control to data, you use the Data Configuration Wizard. Typically, you launch that by opening the control's Common Tasks menu, clicking Choose Data Source, and then choosing New Data Source. Figure 12-1 shows an example: I've dragged a `DataList` control from the Toolbox onto a page, and am about to bind it to a data source.

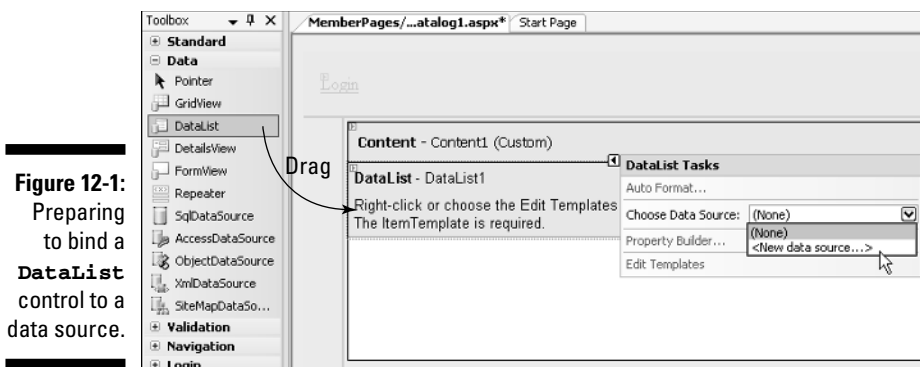


Figure 12-1:
Preparing
to bind a
`DataList`
control to a
data source.

Specifying a data source and connection

The first wizard page asks for the type of data source you want to get data from. Choose Database to get data from a SQL Server database. A name for

that source, usually `SqlDataSource1`, appears in the text box (as shown in Figure 12-2).

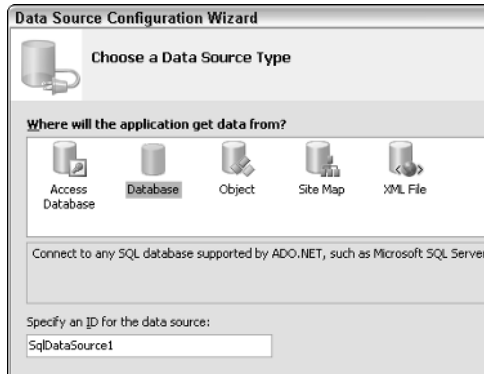


Figure 12-2:
Choosing
SQL Server
as the data
source.

You don't need to change or worry about the name that appears in the text box, it will just be the control's name within the current page. You can click Next to move on.

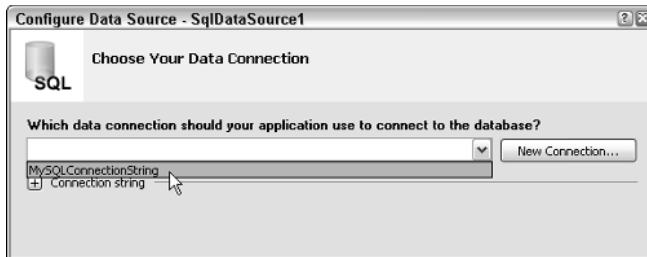
The next wizard page asks you to choose a connection. Typically, the connection to your database has already been defined in your `Web.config` file, so you should always use that one. If that's the case, just choose that existing connection string from the drop-down list, as shown in Figure 12-3.

Defining the connection string

The first time you add a data control to a page, there may not be a name to choose from in the drop-down list. If that's the case, click New Connection. Then in the dialog box that opens, click Browse and navigate to the folder that contains the database file. For example, if you saved your site as `MyVWDSite` in your My Documents folder, you'll need to open `MyVWDSite` in the My Documents folder, then the `App_Data` folder, and finally click on `ASP-NETDB.MDF`, and click Open. Then click OK.

A second page will ask if you want to save the connection string in your Application Configuration file. Choose Yes, and give the connection string a name. For example, I named mine `MyConnectionString`. After you've defined a connection string, there's no need to create others. Each time you add a Data control to a page, you can use the same connection string. That's because the connection string just tells Visual Web Developer where the database (.mdf file) is located. It doesn't specify any particular tables within that database.

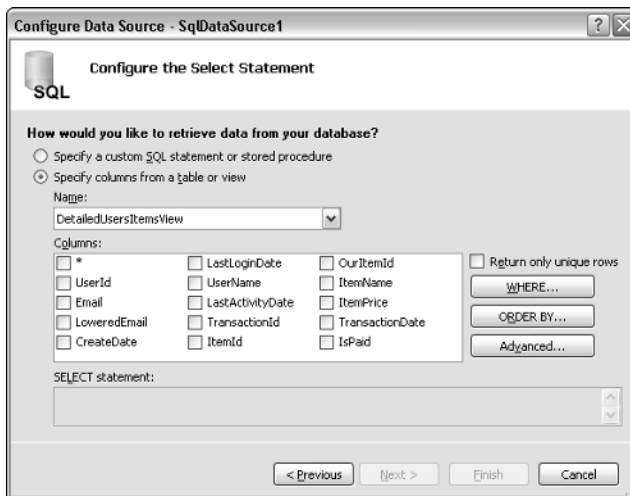
Figure 12-3:
Once you
create a
connection
string, use it
every time.



Configuring the Select Statement

The next wizard page, titled Configure the Select Statement (and shown in Figure 12-4), is where you tell the wizard exactly what you need from the database. Here you specify what you want the control to show on the page. There are two ways you can go about doing that:

Figure 12-4:
The
Configure
the Select
Statement
page.



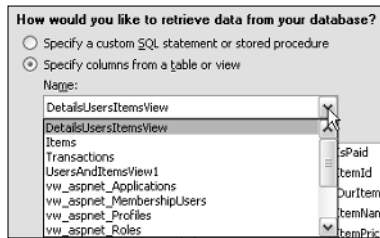
✔ **Specify a Custom SQL Statement or Stored Procedure.** This is the option to choose if you're creating a complex query with multiple connected tables involved. It takes you to the Query Builder, where you can create the SQL statement using the same technique used at the end of Chapter 11 to construct views. If this approach looks a bit intimidating, don't worry; it's rarely (if ever) actually necessary.

✔ **Specify Columns from a Table or View.** This is the cushy (and more commonly used) option. You can easily grab what you want from wherever it is without going through the whole Query Builder rigmarole.

Choosing a table or view

If you choose Specify Columns from a Table or View, the next step is to tell the wizard which table or view contains the data to bind to. You can bind to any table or view you created yourself. If you need data from the membership system, you can bind to any of the `vw_aspnet_ views` in the Name drop-down list shown in Figure 12-5.

Figure 12-5:
Choose whichever table or view contains the data that the control needs.



Choosing columns to retrieve

After you choose a table or view, names of columns (fields) in that table or view appear below, each with a check box. That's where you specify which column (or columns) contain the data you need. To get all columns, choose the * check box. But if you don't need all the columns, then choose only those columns you do need.

For example, let's say I chose `Items` as the table to retrieve data from. I really don't need to be broadcasting my database's primary keys to the general public via the Internet, so I chose to include every column except `ItemID`, as shown in Figure 12-6.



While you're choosing options, you're actually creating the SQL statement that appears below the column names.

To see what the SQL statement you've created so far will retrieve from the database, click Next, then click Test Query. The results of executing the SQL statement appear in a window. Using the options in Figure 12-6 as an example, the Test Query results display all the records from my `Items` table, minus the `ItemID` field, as shown in Figure 12-7. (Note that this is exactly the data needed to show a product catalog on a Web page.)

Figure 12-6:
A request
for all
columns
except
ItemId
from the
Items
table.

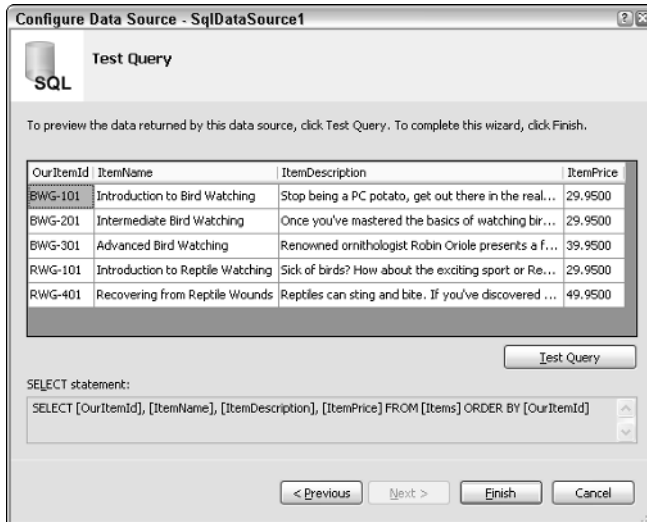
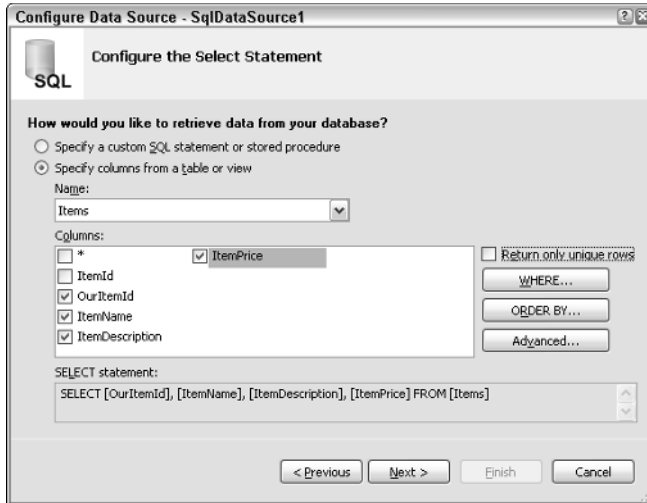


Figure 12-7:
Results of
testing the
query from
Figure 12-6.



As always, do not be concerned about how the data looks, or the arrangement of the columns at this stage. Here you're just saying *what* you want. You'll deal with how it looks on the page later.

If the results you get from your query are not what you expected, no big deal. Just click the Previous button to return to the Configure the Select Statement page and work on the query some more.

If you need data from multiple related tables, and have already created a view that connects the tables, then you can choose that view as your source. For example, if you need information about transactions, you could choose one of the views described at the end of Chapter 11, then specify which columns of information you need about each transaction, as shown in Figure 12-8.

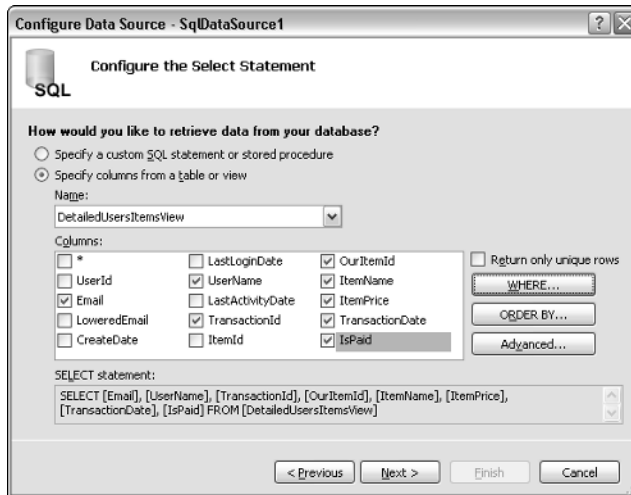


Figure 12-8:
Choosing
columns to
display from
a view.

Testing that query reveals a set of records in which each record represents a transaction made to date. Each transaction would include the name and e-mail address of the user who made the purchase, as shown in Figure 12-9.

Email	UserName	TransactionId	OurItemId	ItemName	ItemPrice	TransactionDate	IsPaid
bob@evoed.com	Bob	20000	BWG-201	Intermediate Bird Watching	29,9500	6/15/2006	<input checked="" type="checkbox"/>
bob@evoed.com	Bob	20001	BWG-301	Advanced Bird Watching	39,9500	6/21/2006	<input checked="" type="checkbox"/>
bob@evoed.com	Bob	20002	RWG-401	Recovering from Reptile Wounds	49,9500	6/25/2006	<input checked="" type="checkbox"/>
anna@coolherds.com	Newbie	20003	BWG-301	Advanced Bird Watching	39,9500	6/3/2006	<input checked="" type="checkbox"/>
eks@EvoEd.com	Eeks	20004	BWG-201	Intermediate Bird Watching	29,9500	6/17/2006	<input checked="" type="checkbox"/>
eks@EvoEd.com	Eeks	20005	BWG-301	Advanced Bird Watching	39,9500	6/21/2006	<input checked="" type="checkbox"/>
carol@evoed.com	Carol	20006	BWG-301	Advanced Bird Watching	39,9500	6/1/2006	<input checked="" type="checkbox"/>
carol@evoed.com	Carol	20007	RWG-101	Introduction to Reptile Watching	29,9500	6/2/2006	<input checked="" type="checkbox"/>
carol@evoed.com	Carol	20008	RWG-401	Recovering from Reptile Wounds	49,9500	6/20/2006	<input checked="" type="checkbox"/>
alice@evoed.com	Alice	20009	BWG-301	Advanced Bird Watching	39,9500	7/1/2006	<input checked="" type="checkbox"/>
alice@evoed.com	Alice	20010	RWG-101	Introduction to Reptile Watching	29,9500	7/21/2006	<input checked="" type="checkbox"/>
alice@evoed.com	Alice	20011	RWG-401	Recovering from Reptile Wounds	49,9500	7/27/2006	<input checked="" type="checkbox"/>

Figure 12-9:
Results of
executing
the query in
Figure 12-8.

Specifying a sort order

By default, records your query retrieves will be listed in the same order they're arranged in the table. If you want a different sort order — say,

alphabetical by name, oldest to newest, or something like that — click the ORDER BY button. The Add ORDER BY Clause dialog box shown in Figure 12-10 opens.

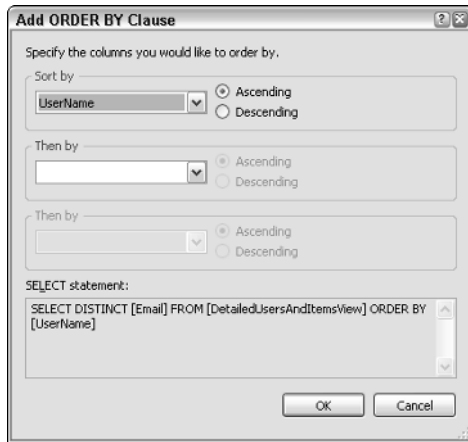


Figure 12-10:
The Add
ORDER BY
Clause
dialog box.

In this dialog box, you can choose how you want records in the query results sorted. For example, if the table or view contains a `UserName` field, you could choose `UserName` and `Ascending` to alphabetize rows by user name. If the table or view contains a `TransactionDate`, and you want to list rows from newest to oldest, choose `TransactionDate` as the Sort by column and `Descending` as the order.

As always, you can choose multiple columns to create sorts within sorts. When you've finished choosing your sort column(s) and orders, click OK. The `SELECT` statement you're creating gains an `ORDER BY` clause to order records by whatever columns you specified.

Showing only unique values

Sometimes you want to extract unique records from a table, so there is no duplication of information in the data returned. That's often the case when you want to use the query results as menu items in a drop-down list.

For example, suppose you want to choose a user (to view their transactions) from a drop-down menu. The menu shouldn't contain all user's names, only users who have records in the `Transactions` table.

A drop-down list usually only needs one or two columns, in this case they would most likely be `UserName` or `Email`, or both. In Figure 12-11, I opted to retrieve only the `Email` and `UserName` columns from the view.

Figure 12-11:
This option, when checked, shows only unique records.

Specify columns from a table or view

Name: DetailedUsersItemsView

Columns:

* LastLoginDate OurItemId Return only unique rows

UserId UserName ItemName

Email LastActivityDate ItemPrice

LoweredEmail TransactionId TransactionDate

CreateDate ItemId IsPaid

WHERE...

ORDER BY...

Advanced...

SELECT statement:

```
SELECT DISTINCT [UserName], [Email] FROM [DetailedUsersItemsView] ORDER BY [UserName]
```

If it were not for the selected check box (Return Only Unique Rows) in Figure 12-11, the query in Figure 12-11 would return a row for every transaction to date, repeating the UserName each time, as shown in the left side of Figure 12-12. Some names are duplicated because any one user might have many transactions. Choosing Return Only Unique Rows eliminates duplicates from the returned rows, leaving only a list of unique names as shown in the right side of Figure 12-12.

Figure 12-12:
Showing duplicate and then unique values.

All rows		Only unique rows	
UserName	Email	UserName	Email
Alice	alice@evoed.com	Alice	alice@evoed.com
Alice	alice@evoed.com	Bob	bob@evoed.com
Alice	alice@evoed.com	Carol	carol@evoed.com
Newbie	anna@coolnerds.com	Eeks	eks@EvoEd.com
Bob	bob@evoed.com	Newbie	anna@coolnerds.com
Bob	bob@evoed.com		
Bob	bob@evoed.com		
Carol	carol@evoed.com		
Carol	carol@evoed.com		
Carol	carol@evoed.com		
Eeks	eks@EvoEd.com		
Eeks	eks@EvoEd.com		

Retrieving specific rows (filtering)

More often than not, you want to bind a data control to *some* of the records in a table or view but not to *all* those records. For example, you might want to extract records only for user Bob or Alice from the view to show that person what products they've purchased. Or, you might want to see a list of everyone who purchased a specific item rather than a list of every transaction ever made.

Limiting the records you retrieve from a table or view is called *filtering* the rows. To specify how you want rows filtered, click the WHERE button on the Configure the Select Statement page. The Add WHERE Clause dialog box opens.

The first step is to choose which column you want to filter on. For example, if you want to extract rows for a particular user only, choose `UserName` or `UserId`. If you want to retrieve rows for a specific item, choose `ItemId`.

After you've chosen a column for the filter, choose an operator. All the usual suspects are available and summarized in Table 12-1.

<i>Operator</i>	<i>Meaning</i>
=	Equals
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Does not equal

After you've chosen the column name and operator, specify where the value to compare to will come from. You use the Source drop-down list for that. It offers several options, but you're most likely to use these two:

- ✓ **Control:** Choose this if the value you want to compare to will come from some other control on the same page, such as a drop-down list.
- ✓ **Profile:** Choose this if the criterion for selecting records has anything to do with choosing only records that are relevant to the specific user who is viewing the page.



It's unlikely that you'd ever have to choose `Cookie`, `Form`, `QueryString`, or `Session` from the Source drop-down list. You should be able to do anything needed here if you use either the `Control` or `Profile` source.

What happens next depends on what you chose in the steps leading up to this point. You'll see more information and examples shortly. But for the sake of example, let's say you chose `UserName` as the column to query, `=` as the operator, and `Profile` as the Source. The next step would be to specify the property name which, in this case, is `UserName` (same as the column name). Figure 12-13 shows an example in which only records that have the current user's name in the `UserName` column of the table will be extracted from the table.

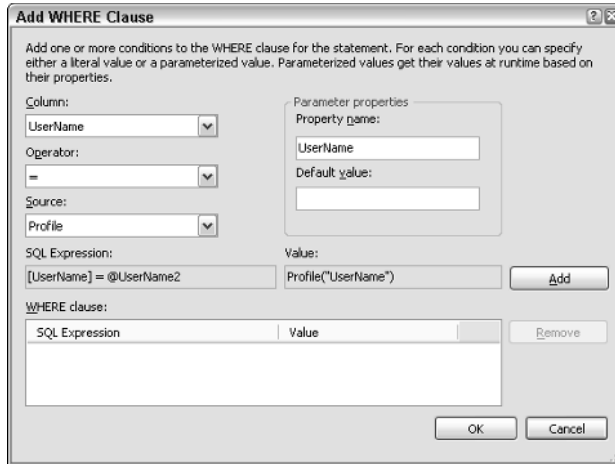


Figure 12-13:
Retrieve records for the current user only.



When you've specified your criterion, you must click the Add button to go on. (If you forget this step, the query won't work and you'll drive yourself nuts trying to figure out what's wrong.) When you click Add, your current selections are converted to SQL down at the bottom of the dialog box. The other controls are cleared so you can create additional criteria, if need be. If you don't need to add any more criteria, then click OK to return to the previous wizard page.

The `SELECT` statement you're creating now contains a `WHERE` clause that looks like this:

```
WHERE ([UserName] = @UserName)
```

That's SQL for "get records only for whichever user requested this page."



It's important to understand that the `WHERE` clause you create must refer to columns in the current query. The query shown here wouldn't work in the `Items` table because there's no `UserName` column in the `Items` table. But it works in `DetailedUsersItemsView` because that view *does* contain a `UserName` column.

When you test a query that includes a `WHERE` clause, you'll need to manually type in some plausible data to verify that the `SELECT` statement works. This means click Next, and then click Test Query. You won't see query results immediately; instead, you see the Parameter Values Editor dialog box shown in Figure 12-14.



A parameter is a chunk of information being passed to the SQL statement. For example, we can feed the `WHERE` clause the parameter `Bob`. In return, the SQL statement spits back only records that have `Bob` in the `UserName` column.

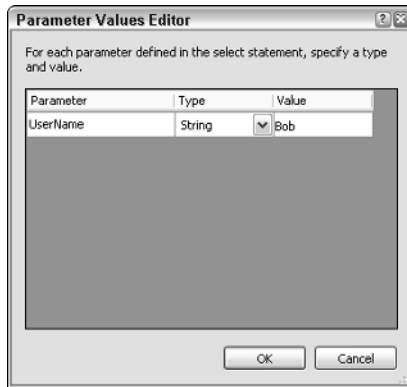


Figure 12-14:
The
Parameter
Values
Editor dialog
box.

To test the query, you must enter a realistic value into the Value box in the query editor. For example, I know for a fact that I have a user named Bob in my current membership system. So I could type that name into the Value box, (refer to Figure 12-14).

After you enter a test value and click OK, the query executes. In this example, only rows that have *Bob* in the `UserName` field are retrieved from the view, as shown in Figure 12-15.

Figure 12-15:
Only rows
for user Bob
are returned
by the
SELECT
statement.

Email	UserName	TransactionDate	IsPaid	OurItemid	ItemName	ItemPrice
bob@evoed.com	Bob	6/25/2006	<input checked="" type="checkbox"/>	RWG-101	Recovering from Reptile Wounds	49.9500
bob@evoed.com	Bob	6/21/2006	<input checked="" type="checkbox"/>	BWG-301	Advanced Bird Watching	39.9500
bob@evoed.com	Bob	6/15/2006	<input checked="" type="checkbox"/>	BWG-201	Intermediate Bird Watching	29.9500

The query results show every transaction made by user Bob. When you bind this to a data control on your page, however, the query won't show *Bob's* record on the page. It will show records for whatever user requested the page.

Given those options you just discovered, you can retrieve anything you want from your database, no matter how large or small. When you've accurately defined the columns and rows that your data-bound control needs, click Next. You're taken to the Test Query page again. There you can click the Test Query button once to make sure your query is retrieving the data you need. Then click Finish when you're certain the query provides the data your control needs.



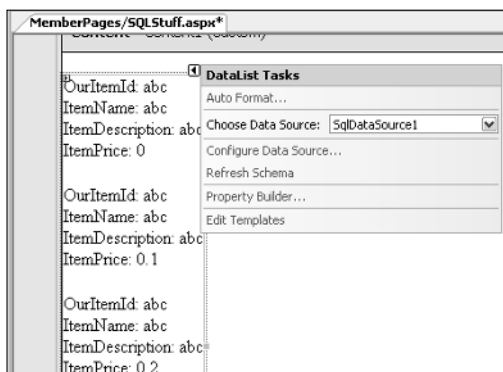
If you tested a query previously, the results that show up on the Test Query page will be the ones left over from that previous text. To see what your current query produces, you must click the Test Query button.

Clicking Finish closes the wizard and brings you back to the Design view of the page. The data-bound control will probably look completely different (though still random and ugly, because it's only a placeholder for data to be provided later).

Data controls in Design view

In Design view, data controls don't look like much. The control shows column names from the underlying table or view. For example, Figure 12-16 shows a `DataList` control after binding it to the `Items` table (without the `ItemId` column). Instead of showing actual data from the table, the control just shows placeholders like `abc` for text and `0.1` for numbers.

Figure 12-16:
A
`DataList`
control after
binding to
columns
from the
`Items`
table.



To see what the data-bound control will display to people accessing your Web site normally, view the page in a Web browser. The placeholder text is replaced with actual text from the table, as shown in Figure 12-17. (There's more text than can be seen without scrolling. But trust me, the page shows all records from the `Items` table.)

Keep in mind that Figure 12-17 is only an example. There are many data controls to choose from, and an infinite number of ways you can display data on a page. But the general procedure of going through the Data Configuration Wizard is the same, regardless of what data control you use or how you format your data.

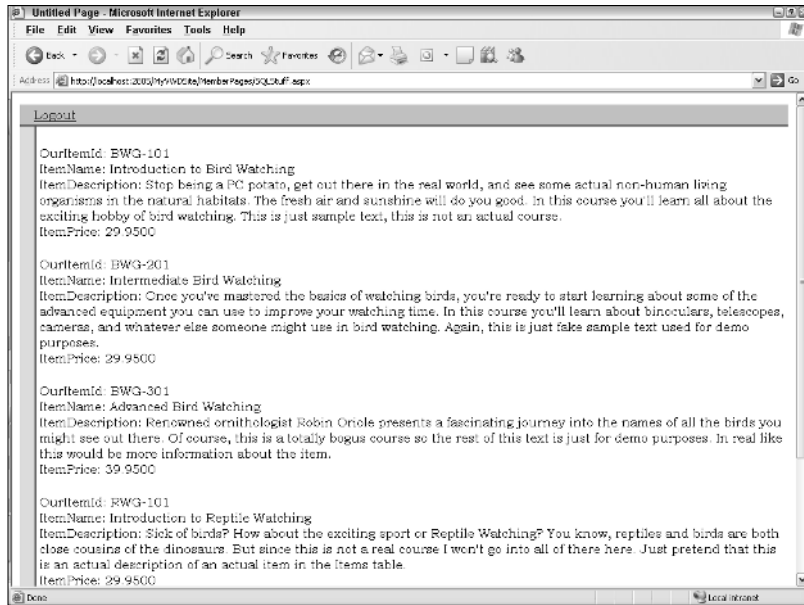


Figure 12-17:
The
page from
Figure 12-6
as seen
in a Web
browser.

If you ever change your mind about the columns and rows you chose for a data-bound control, it's no big deal. Just click the control's Common Tasks menu and choose Configure Data Source. Click Next on the first wizard page, and you'll be taken back to the Configure SQL Statement page, where you can change which columns are retrieved, the sort order, the WHERE clause, or any combination of those.

Formatting Dates and Numbers

Unless you specify otherwise, data from SQL Server tables look on a page as they do in a table, which means money fields display in the format 29.9500 and date/times appear in the format 6/15/2006 12:00:00 AM. Of course, you're not stuck with those formats.

Exactly how you change the format of a date or number varies from one data control to the next. (You'll see examples in upcoming sections.) But the symbols you use to format dates and times are always the same.

The full set of things you can do, formatting-wise, are all documented in the .NET Framework and C# documentation under the general moniker of *composite formatting*. The ones you're most likely to actually use in a Web site are summarized in Table 12-2. The first one, {0:C} or {0:c}, works with any field that's the money or smallmoney data type. The others work with fields of the datetime and smalldatetime data type.

<i>Symbol</i>	<i>Name</i>	<i>Sample Output</i>
{0:C} or {0:c}	Currency	\$29.95
{0:d}	Short date	6/1/2006
{0:D}	Long date	Thursday, June 01, 2006
{0:t}	Short time	12:00 AM
{0:T}	Long time	12:00:00 AM
{0:f}	Full (short time)	Friday, June 02, 2006 12:00 AM
{0:F}	Full (long time)	Thursday, June 01, 2006 12:00:00 AM
{0:g}	General (short time)	6/1/2006 12:00 AM
{0:G}	General (long time)	6/1/2006 12:00:00 AM

Some Security Considerations

In any given database, there is sure to be information that users should never see. There will also be much information that users are allowed to see but not change. It's up to you to decide which is which — and to provide all the necessary security.

A relatively simple way to deal with this is to create a new folder for pages that no user (other than you) can see. Here's how:

- 1. Right-click the site name at the top of Solution Explorer.**
- 2. Choose New Folder.**
- 3. Name this folder AdminPages.**
- 4. Using the Web Site Administration Tool, create a new role, perhaps named Admin.**
- 5. Add a new access rule that allows people in the Admin role to access pages in AdminPages, and denies access to both anonymous users and site members.**
- 6. Finally, create a new user account for yourself and put yourself in both the Admin and SiteMembers roles.**

You'll need to log in to that new user account before you can view any pages you put into the new AdminPages folder.



If you don't have the slightest idea what I'm talking about here, see Chapter 3.

Using the GridView Control

As its name implies, the `GridView` control shows data from a table or view in a grid consisting of rows and columns. As always, you can specify exactly which columns and rows (and from which tables or views) in your database the grid should show. The `GridView` can be used both to display data, as well as to add, change, and delete data in a table.

If your intent is to show data to users in the `GridView` control, then you can bind the `GridView` to any table or view in your database. If you want to use the `GridView` to edit data in the table, then you cannot bind the control to a view. (Data from views can never be edited by the user.) Furthermore, the `GridView` must be bound to one table only, and that table must contain a primary key field.

An instant GridView control

If you want to use a `GridView` control to show all columns and rows from a single table in your database, your job is easy. Here's what you should do:

1. In Database Explorer, click the + sign (if necessary) to expand the tables list.
2. Drag the name of a table that you created (not one of the `aspnet_*` tables) onto the page.

For example, Figure 12-18 shows the results of dragging a table named `Items` into the content area of a page that has a Master Page.

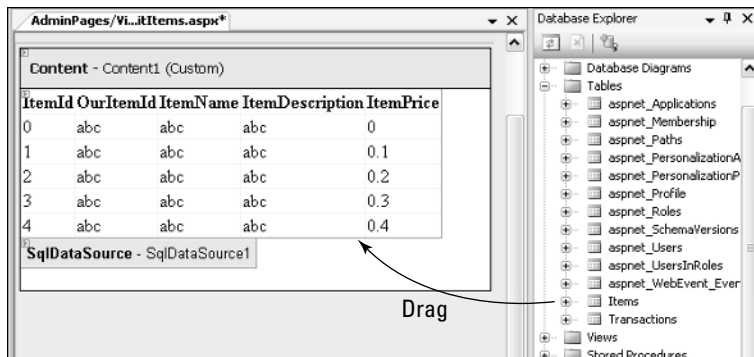


Figure 12-18: Drag a table to a page to create an instant GridView.

If you want to show something in a `GridView` other than the contents of a single table, you can create a `GridView` control and bind it to appropriate data using the Data Configuration Wizard. Here are the steps:

1. In the Toolbox, expand the Data category of controls.
2. Drag a `GridView` control from the Toolbox onto your page.
3. From the `GridView`'s Common Tasks menu, select Choose Data Source → <New data source>.
4. Use the Data Source Configuration Wizard to specify which data you want the control to show.

When you complete the wizard and click Finish, the `GridView` control shows a column heading for each column you specified in the Data Source Configuration Wizard.

In a Web browser, the data will look something like Figure 12-19 (assuming you don't do any formatting). The control actually shows all records from the table. The figure only shows a few records.

Logout				
ItemId	OurItemId	ItemName	ItemDescription	ItemPrice
10001	BWG-101	Introduction to Bird Watching	Stop being a PC potato, get out there in the real world, and see some actual non-human living organisms in the natural habitats. The fresh air and sunshine will do you good. In this course you'll learn all about the exciting hobby of bird watching. This is just sample text, this is not an actual course.	29.9500
10002	BWG-201	Intermediate Bird Watching	Once you've mastered the basics of watching birds, you're ready to start learning about some of the advanced equipment you can use to improve your watching time. In this course you'll learn about binoculars, telescopes, cameras, and whatever else someone might use in bird watching. Again, this is just fake sample text used for demo purposes.	29.9500
10003	BWG-301	Advanced Bird Watching	Renowned ornithologist Robin Oriole presents a fascinating journey into the names of all the birds you might see out there. Of course, this is a totally bogus course so the rest of this text is just for demo purposes. In real like this would be more information about the item.	39.9500
10004	RWG-101	Introduction to Reptile Watching	Sick of birds? How about the exciting sport or Reptile Watching? You know, reptiles and birds are both close cousins of the dinosaurs. But since this is not a real course I won't go into all of there here. Just pretend that this is an actual description of an actual item in the Items table.	29.9500

Figure 12-19:
The page
from
Figure 12-18
in a Web
browser.

Formatting the GridView control

The default appearance of a `GridView` control isn't necessarily pretty. But you have lots of options for making it look and act the way you want. Most of these options are available from the control's Common Tasks menu shown in Figure 12-20.

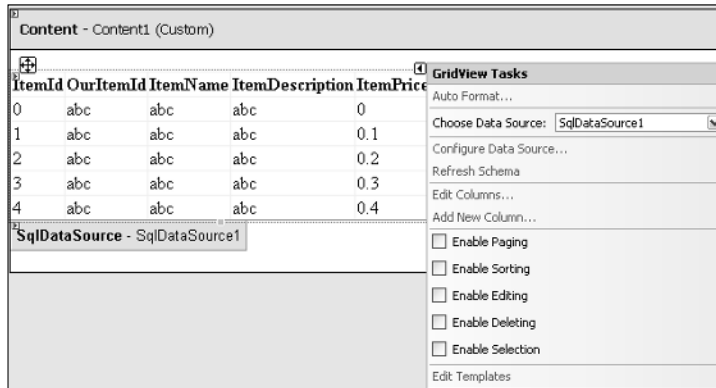


Figure 12-20:
Common
Tasks menu
for a
`GridView`
control.



The gray `SqlConnection` box is just a placeholder that defines the control's data source, and won't show up on your Web page. You can show or hide those gray boxes by choosing `View` ⇄ `Non Visual Controls` from VWD's menu.

As with most controls, the Common Tasks menu for the `GridView` has an `Auto Format` option that you can choose to apply a predefined formatting style to the control. Other major items on the menu are as follows:

- ✓ **Enable Paging:** Adds a navigation bar to the bottom of the `GridView`, allowing users to page through multiple records. Useful for displaying data from larger tables to prevent all records from showing up on a single page.
- ✓ **Enable Sorting:** Converts each column title to a clickable link. In the browser, users can click any column heading to sort the rows by that column.
- ✓ **Enable Editing:** Allows users to change any value in any row or column. This option is only available if the control is bound to a single table that has a primary key.
- ✓ **Enable Deleting:** Allows users to delete records. This option is only available if the control is bound to a single table that has a primary key.



If you enable editing or deleting, that means anybody who views the page can change or delete records in the table. If you don't want users to

do that, be sure to put the page that contains the control into a folder that users can't access.

- ✓ **Enable Selection:** Allows users to select a record. That record, in turn, can be used as a filter for other data controls on the same page. You'll see an example in the section titled "Creating Master-Details Forms."

Formatting GridView dates and times

If your `GridView` control shows currency or date/times, you use the Edit Items option on the Common Tasks menu to format those columns. After you choose Edit Items, you'll be taken to a dialog box titled Fields. Under Selected Fields in that dialog box, click the name of the field to which you want to apply a format. The Properties box for the `BoundField` control then shows properties for that selected field.

Scroll down through the properties until you get to the `DataFormatString` property and then enter your formatting code. Figure 12-21 shows an example in which I've applied the `{0:c}` formatting code to the `ItemPrice` control.

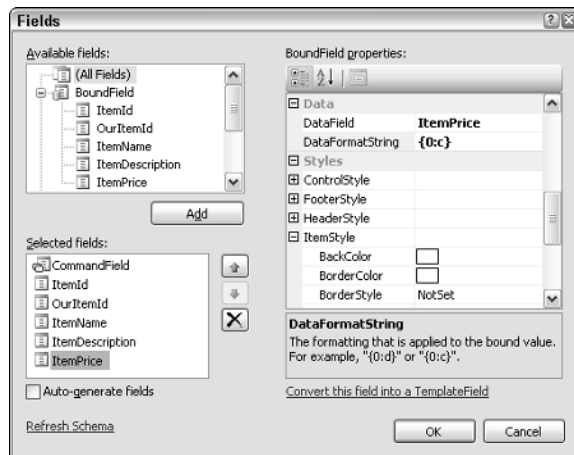


Figure 12-21:
Fields dialog
box for a
`GridView`
control.

To right-align or center text within a column, scroll down a bit further in the Properties sheet and expand the `ItemStyle` category. Then set the `HorizontalAlign` property to `Right` or `Center`.

Arranging and hiding columns

To arrange or hide columns in a `GridView`, use the arrow and Delete buttons just to the right of the Selected Fields list. For example, to move a column to the left within the `GridView`, click its name, then click the Up button.

To hide a column from `GridView` display, click its name and click the black X button to the right of the Selected Fields list.



If your `GridView` supports editing, deleting, or selecting, the Selected Fields list will include an item named `CommandField`. That item represents the left-most column in the table where [Edit](#), [Delete](#), and [Select](#) links are placed.

If you don't want a particular column to show up in the `GridView`, just remove its name from the list of Selected Fields. For example, I could remove the `ItemId` field from the current `GridView` control by clicking `ItemId` in the Selected Fields list and then clicking the black X button just to the right of the list.

When you've finished formatting fields in the Fields dialog box, click OK to save your changes and return to the page. To get an accurate picture of how your selections will play out, be sure to view the page in a Web browser.

Styling the whole GridView

You can style the `GridView`, as a whole, much as you would any other item. Right-click the control, choose Style, and use the Style Builder to define the style.

You can also style the control via its Properties sheet. The Properties sheet for `GridView` provides extremely fine-grained control over the exact appearance and behavior of every `GridView` nook and cranny. For example, if you've enabled `Paging` in your control and want to set the number of rows that appear on each page, change the `PageSize` property under `Paging` to however many records you want each page to show.

If you want to apply a consistent look and feel to `GridView` controls used throughout your site, consider creating a CSS class. For example, you could create a CSS style rule named `GView` that looks like this:

```
.GView
{
    border-right: #191970 thin solid;
    border-top: #191970 thin solid;
    font-size: smaller;
    border-left: #191970 thin solid;
    color: #483d8b;
    border-bottom: #191970 thin solid;
    border-collapse: collapse;
    background-color: #f0f8ff;
}
```

Binding to DropDownList Controls

Even though a `DropDownList` control isn't a data control, per se, you can bind data from a database to that control. This is especially useful when the

drop-down list needs to show current data from the database — for example, all users, all items in an `Items` table, or all users who have made purchases. Whatever you choose from the drop-down list can then be used as a filter for specifying rows to display in a nearby data-bound control.

For example, suppose you want to create a page in which you can choose any user in your database and see his or her transactions and profile properties (or whatever) only. Your first move is to create some means of choosing one user. A drop-down list might work nicely for that.

To create a drop-down list that shows current database data, follow these steps:

- 1. Create or open the .aspx page on which you want to place the control.**
- 2. From the Standard category in the Toolbox, drag a `DropDownList` control onto the page.**

By default, the control is named `DataList1` (assuming it's the first `DropDownList` control you added to the page). It's important to know the name of the control because to use the control's value for anything useful later, you must refer to it by exactly that name.

- 3. On the `DataList` control's Common Tasks menu, select **Enable Postback**.**

The above step is important if you intend to use the drop-down list as a means of filtering rows in a table.

- 4. From the Common Tasks menu, select **Choose Data Source**.**
- 5. On the wizard page that opens, choose **New Data Source**.**
- 6. On the first wizard page, choose **New Data Source** from the drop-down list.**
- 7. Go through the usual steps in the first wizard pages (that is, choose **Database**, click **OK**, choose your usual connection string, and click **Next**).**

At this point, you're in the `Configure SQL Statement` page.

- 8. Choose the table or view that contains the columns you want to show in the drop-down menu.**

For example, if you want the drop-down menu to show an alphabetical list of all current users, you could choose `vw_aspnet_MembershipUsers`.

- 9. Choose the column (or columns) you want to use for the drop-down menu.**

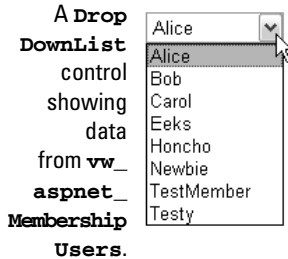
For example, choose `UserName` to make the drop-down list show a list of user names.

- 10. Use the **ORDER BY** button to sort the items into alphabetical order.**

When you've finished, you can click `Next` (as necessary) and then `Finish` to work your way back to the control.

To test the drop-down list, you have to view the page in a Web browser. Using the example in which I opted to show the `UserName` and `Email` columns from the `vw_aspnet_MembershipUsers` view, my drop-down list looks like the one shown in Figure 12-22.

Figure 12-22:



Using a DropDownList to filter records

To use a `DropDownList` control to filter records in another data-bound control (such as a `GridView`), several steps are required. I'll use a `GridView` control to explain the basic process, but the same idea works with other `Data` controls described later.

The first step is to add the control to the page and bind it to the values you want the control to show. For example, suppose that after you choose a user name from the drop-down list, you want to see all the transactions that user has made. You could get that result by dragging a `GridView` control to the same page as the `DropDownList` control and using the drop-down list control to filter the records it displays.

Any time you add a second `Data` control to a page, you want to be sure to choose `<New Data Source...>`, and not try to use the same data source that the first control uses. That's because the data source defines the exact rows and columns that the control will show. And the rows and columns the second control shows will be different from what the first control shows. By default, the new data source is named `SqlDataSource2`.

When you choose a connection string, you want to reuse the same connection string you use for everything else. That's because the connection string only tells the control where the database is located — it doesn't specify any tables or views within the database.

When you're choosing a table, view, or column, choose one that has the same column as the `DropDownList` control; `UserName`, in this example. That's because rows to be retrieved need to be filtered by values in that column name.

The `DetailedUsersItemsView` view (described in Chapter 11) contains lots of information about transactions, and includes a `UserName` field, so you could choose that as the view. Then, just choose the columns you want the `GridView` control to show, as shown in Figure 12-23. You can also choose a sort order using the `ORDER BY` button.

Figure 12-23:
Columns to
show in the
GridView
control
selected.

The critical step in the filtering process is limiting records to those that match the name in the `DropDownList`. Click the `WHERE` button to get started on that. Then set your options as follows:

- ✓ **Column:** The name of the column that contains the value needed for filtering; `UserName` in this example.
- ✓ **Operator:** This is typically the `=` operator, but it can be any available operator.
- ✓ **Source:** Where the value to be searched for comes from. In this example, that would be `Control` because the value to look for is in a `DataList` control.
- ✓ **Control ID:** The name of the control that contains the value to look for; `DropDownList1` in this example.

So in this example, the selections would look like Figure 12-24. Those selections are just a graphical way of saying “Retrieve only rows in which the `UserName` column’s value equals the value of the control named `DropDownList1`.”

You must remember to click `Add`, at which point the selections get translated into SQL. Click `OK`, click `Next`, and then click `Finish` to work your way back to the page.

To test the page, view it in the browser. Initially you’ll see just the `DropDownList` control. To view any user’s transactions, select that user’s name from the drop-down list. The `GridView` control “re-filters” each time you make a selection, so you can check out different users. Figure 12-25 shows an example in which Carol is selected in the `DropDownList` control, so only her transactions are visible in the `GridView` control below the `DropDownList` control.

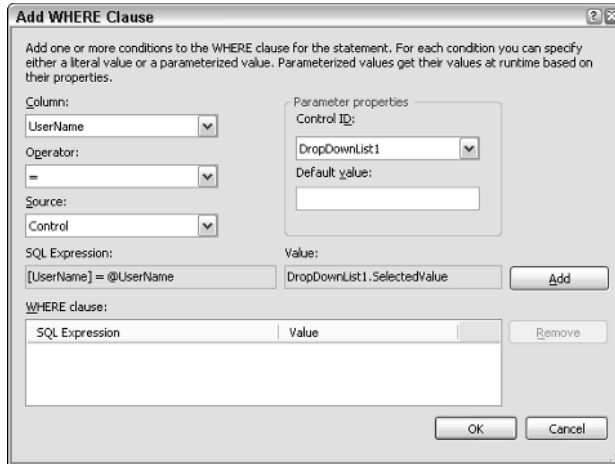


Figure 12-24:
Retrieving
only the
records that
match the
UserName
in **Drop
DownList1**.

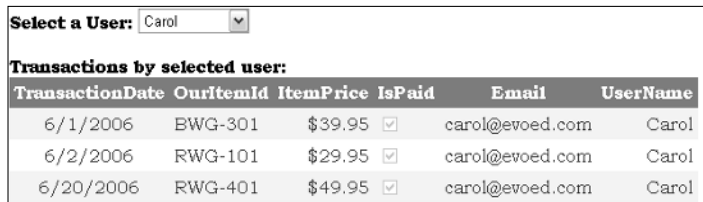


Figure 12-25:
GridView
control,
showing
only the
records for
selected
user Carol.

In Figure 12-25, I gussied up the `GridView` control a bit, using techniques described in the preceding section. I also typed some text directly onto the page to clarify what's going on.

Viewing and editing user properties

If you added profile properties to your site, sometimes you want to see and edit the properties of your user profiles. The tricky part is that you can't use the normal syntax such as `Profile.propertyName` to refer to a specific user's properties. That's because that standard syntax always refers to whoever is viewing this page. When *you're* viewing the page, that's always going to be you (well, yeah), so you'd see only your own profile properties.

To see the properties of some other user, use the following syntax instead:

```
Profile.GetProfile(UserName).propertyName
```

where *UserName* is the name of the user for which you want to view a profile, and *propertyName* is the name of the property. If the *UserName* value is stored in a control, like `DropDownList1`, then *UserName* is actually the value of that control, as given here:

```
Profile.GetProfile(DropDownList1.SelectedValue).propertyName
```

So, if the name Bob is currently selected in the control named `DropDownList1`, then `Profile.GetProfile(DropDownList1.SelectedValue).FirstName` refers to user Bob's `LastName` property.

Given that, you could add a table to the page that contains a `Textbox` control for each profile property you want to display. For example, Figure 12-26 shows some `Textbox` controls, and their names, in a page with a drop-down list control on it. I've also pointed out the names of the drop-down list control (`DropDownList1`) and the button (`Button1`), as those names are important programmatically.

Figure 12-26:
Text box controls that can be used to show profile properties.

Selected User's Profile Properties	
First Name:	<input type="text"/>
Last Name:	<input type="text"/>
Address 1:	<input type="text"/>
Address 2:	<input type="text"/>
City:	<input type="text"/>
State/Province:	<input type="text"/>
ZIP/Postal Code:	<input type="text"/>
Country:	<input type="text"/>
Preferred Theme:	<input type="text"/>
<input type="button" value="Submit Changes"/>	

Each time you choose a name from the drop-down list, the `Textbox` controls need to be populated with properties from the currently-selected user, which means they have to be updated every time the value in the drop-down list changes. To make that happen, double-click the drop-down list control to get

to its `SelectedIndexChanged` event handler. Then type in the necessary code to populate each `Textbox` control. So the whole C# procedure looks like this:

```
//Populate Textbox controls with selected user's profile properties.
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    txtFirstName.Text = Profile.GetProfile(DropDownList1.SelectedValue).FirstName;
    txtLastName.Text = Profile.GetProfile(DropDownList1.SelectedValue).LastName;
    txtAddress1.Text = Profile.GetProfile(DropDownList1.SelectedValue).Address1;
    txtAddress2.Text = Profile.GetProfile(DropDownList1.SelectedValue).Address2;

    txtCity.Text = Profile.GetProfile(DropDownList1.SelectedValue).City;
    txtStateProv.Text =
        Profile.GetProfile(DropDownList1.SelectedValue).StateProvince;
    txtZIPpostal.Text =
        Profile.GetProfile(DropDownList1.SelectedValue).ZIPPostalCode;
    txtCountry.Text = Profile.GetProfile(DropDownList1.SelectedValue).Country;
    txtPrefTheme.Text =
        Profile.GetProfile(DropDownList1.SelectedValue).PreferredTheme;
}
```

To allow editing you'd need to include a button like the `Submit` button in the `EditProfile.aspx` page from Chapter 9. But, again, replace `Profile.` with `Profile.GetProfile(DropDownList1.SelectedValue).` as shown here:

```
//Replace current user's profile properties with contents of Textbox controls.
protected void Button1_Click(object sender, EventArgs e)
{
    Profile.GetProfile(DropDownList1.SelectedValue).FirstName = txtFirstName.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).LastName = txtLastName.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).Address1 = txtAddress1.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).Address2 = txtAddress2.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).City = txtCity.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).StateProvince =
        txtStateProv.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).ZIPPostalCode =
        txtZIPpostal.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).Country = txtCountry.Text;
    Profile.GetProfile(DropDownList1.SelectedValue).PreferredTheme =
        txtPrefTheme.Text;
}
```

Assuming you add that to the previous page example, when you choose a user from the drop-down menu, that page shows all profile properties for that user as well as all of that user's transactions, as shown in Figure 12-27.

Select a User:

Selected User's Profile Properties:

First Name:	<input type="text" value="Robert"/>
Last Name:	<input type="text" value="Wingnut"/>
Address 1:	<input type="text" value="456 Main St."/>
Address 2:	<input type="text"/>
City:	<input type="text" value="Mayberry"/>
State/Province:	<input type="text" value="NC"/>
ZIP/Postal Code:	<input type="text" value="43210"/>
Country:	<input type="text" value="USA"/>
Preferred Theme:	<input type="text" value="Nerdy"/>

Transactions by selected user:

TransactionDate	OurItemId	ItemPrice	IsPaid	Email	UserName
6/15/2006	BWG-201	\$29.95	<input checked="" type="checkbox"/>	bob@evoed.com	Bob
6/21/2006	BWG-301	\$39.95	<input checked="" type="checkbox"/>	bob@evoed.com	Bob
6/25/2006	RWG-401	\$49.95	<input checked="" type="checkbox"/>	bob@evoed.com	Bob

Figure 12-27:
Page shows profile properties and transactions for selected user.

Using the DetailsView Control

The `DetailsView` control in the Toolkit's `Data` category is similar to the `GridView` in that you can use it to show data as well as to edit and delete data. The main difference is that the `GridView` is designed to show multiple rows and columns in a tabular format, and `DetailsView` is designed for working with one record at a time.



For readers who are familiar with Microsoft Access, a `GridView` is like a datasheet and a `DetailsView` control is more like a form. The `FormView` control is an older ASP.NET 1 control that isn't quite as easy to use as the new ASP.NET 2.0 `DetailsView` control.

Binding a DetailsView control

The `DetailsView` control allows you to add, edit, and delete records in a table. However, it can only do so if it's bound to a single table that has a primary key. If you bind a `DetailsView` control to a view or to multiple tables, you can still see data in the control, but you can't edit data in the underlying table(s).

Even if you do bind a `DetailsView` control to a single table that has a primary key, you still won't be able to use it to edit data in the underlying table unless you take some extra steps during the binding process. From start to finish, the steps would be:

1. Drag a `DetailsView` control from the **Data** category of the Toolbox onto the page.
2. On its **Common Tasks** menu, click **Choose Data Source** and choose **<New Data Source...>**. The **Data Source Configuration Wizard** opens.
3. The next steps are the same as always: **Choose Database**, click **OK**, use the same connection string you always use, and click **Next**.
4. If you want to use `DetailsView` to add/edit/delete table records, choose a table that has a primary key.

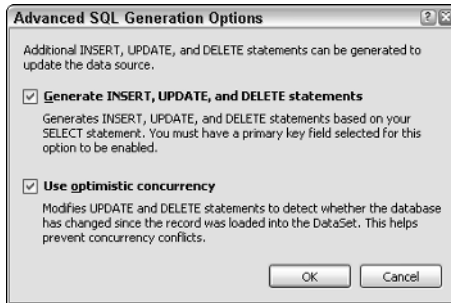
I'll use my sample `Items` table to illustrate.

5. Click ***** to choose **All Columns** (if you intend to use the control to enter/edit/delete records).

Optionally you can use the **WHERE** and **ORDER BY** buttons in the usual manner to limit records the control retrieves to set a sort order.

6. If you want to be able to add/edit/delete records, click the **Advanced** button.
7. In the dialog box that opens, choose both options, as shown in **Figure 12-28**.

Figure 12-28:
Choose both options if you want to use `DetailsView` to modify data.

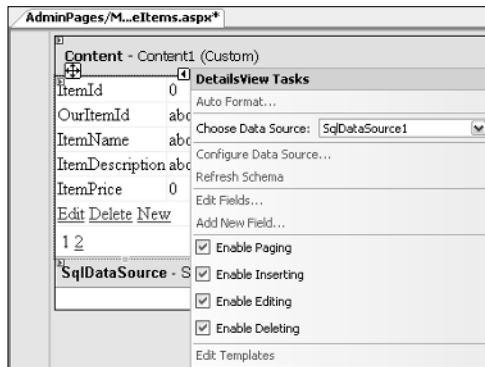


That scary “optimistic concurrency” thing is just a means of dealing with situations in which two people edit a record at the same time. Choosing that option is a good thing because it basically says to the app “*You deal with that headache, so I don’t have to.*”

8. Click **OK**, then click **Next** and **Finish**, as appropriate, to return to the page.

Assuming you didn't skip Steps 6 and 7 above, the Common Tasks menu for the `DetailsView` control shows options for inserting, editing, and deleting records as shown in Figure 12-29. There's also an `Enable Paging` option, which, if selected, lets you page through multiple records in the underlying table.

Figure 12-29:
Common
Tasks menu
for a
`DetailsView`
control.

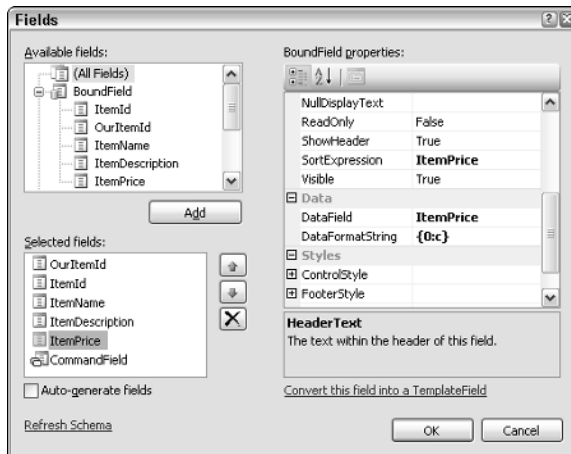


Formatting a `DetailsView` control

Formatting a `DetailsView` control is very similar to formatting a `GridView`. As always, you can use the `Auto Format` option on `Common Tasks` to choose a general appearance for the control. Use the `Edit Fields` option to choose the order of items in the control and currency/date formats.

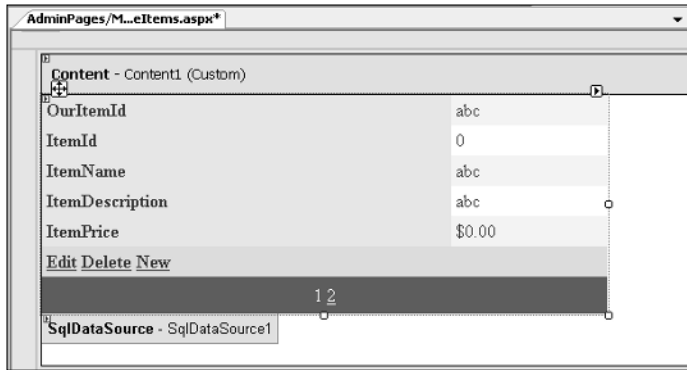
In Figure 12-30, for example, I moved the `OurItemId` control to the top of the list under `Selected Fields`. Then I clicked the `ItemPrice` field name and set its `DataFormatString` to `{0:c}` to display the `ItemPrice` in currency format.

Figure 12-30:
The `Fields`
dialog box
for a
`DetailsView`
control.



You can also size the control by clicking it and dragging any sizing handle that appears in its border. (The widths of the columns in Design view don't accurately match how things will look in the browser, but don't let that bug you.) In Figure 12-31, for example, I've made the `DetailsView` control much wider than it was originally.

Figure 12-31:
The
**Details
View**
control,
widened.



Of course, you never really know how a control will look in real life until you view its page in a browser. Figure 12-32, for example, shows how the control from the previous example looks in Internet Explorer. The little numbers at the bottom represent other records in the same table. (They appear because I chose `Enable Paging` in the control's `Common Tasks` menu.)

Figure 12-32:
**Details
View**
control in
Internet
Explorer.



At the bottom of the control are the `Edit`, `Delete`, and `New` options. Clicking any one of those lets you (respectively) edit, delete, or change the record currently displayed in the control.

Creating Master-Details Forms

You can combine `DropDownList`, `GridView`, and `DetailsView` controls on a single page to create a *Master-Details* form. The `DropDownList` and `GridView` controls are used to zero in on a specific record. The `DetailsView` control then allows you to edit one record. Such a scenario is useful when the database contains a lot of records and finding specific records isn't always easy.

Figure 12-33 shows an example where choosing a user's name from the `DropDownList` control displays detailed information about all transactions made by that user in the `GridView` control to the right. Clicking `Select` at the left side of the `GridView` then displays the actual record from the `Transactions` table that represents that transaction. The `DetailsView` control contains `Edit` and `Delete` options, allowing you to change or delete that one record in the `Transactions` table.

The screenshot shows a web page with three main sections:

- 1. Choose a user:** A dropdown menu with "Bob" selected.
- 2. Select a transaction:** A table with columns: Trans Id, OurItemId, ItemName, Price, Date. It lists three transactions:

Trans Id	OurItemId	ItemName	Price	Date
Select 20000	BWG-201	Intermediate Bird Watching	\$29.95	6/15/2006
Select 20001	BWG-301	Advanced Bird Watching	\$39.95	6/21/2006
Select 20002	RWG-401	Recovering from Reptile Wounds	\$49.95	6/25/2006
- 3. Edit or delete transaction:** A details view for the selected transaction (20001) showing fields:

UserId	572a1ed-3553-40c7-99ce-1ad3bcece831
ItemId	10003
TransactionDate	6/21/2006
IsPaid	<input checked="" type="checkbox"/>
TransactionId	20001
Edit Delete	

Figure 12-33:
Sample
Master-
Details page
for editing
transactions.

Before we get into specifics, let me point out that I started by adding a table with two columns and three rows to a blank page. The text you see on the page, like 1. Choose a user is just text typed right into the table cell. The text and `DropDownList` control are in the top-left cell. The text and `GridView` control are in the cell to the right of that one.

The second row in the table I left empty to put a little empty space above the third row. The left column of the third row contains the text 3. Edit or Delete transaction, and the last table cell contains the `DetailsView` control.

That's just how I organized the text and controls on the page. What makes all the controls work together is the way in which the GridView control is filtered to show only transactions for the user name selected in the drop-down list, and the way the DetailsView control is filtered to show only the record associated with the selected row in the GridView control.

Master-Details DropDownList control

Let's start with the DropDownList control, which is named DropDownList1. To show user names, that control is bound to the UsersAndItemsView view from Chapter 11. From that view, only UserId and UserName are selected. Its Return Only Unique Rows option is selected to show only unique user names. I set its ORDER BY option to sort the records by UserName, so the names will be in alphabetical order by name. Figure 12-34 shows the Configure the Select Statement page for that DropDownList control.

Figure 12-34:
The
Configure
the Select
Statement
page for
Master-
Details
DropDown
List1.

Specify columns from a table or view

Name:
UsersAndItemsView

Columns:

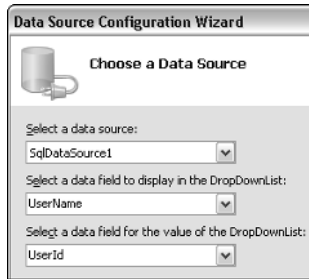
<input type="checkbox"/> *	<input type="checkbox"/> TransactionId	<input checked="" type="checkbox"/> Return only unique rows
<input checked="" type="checkbox"/> UserId	<input type="checkbox"/> ItemId	<input type="button" value="WHERE..."/>
<input checked="" type="checkbox"/> UserName	<input type="checkbox"/> OurItemId	<input type="button" value="ORDER BY..."/>
<input type="checkbox"/> TransactionDate	<input type="checkbox"/> ItemName	<input type="button" value="Advanced..."/>
<input type="checkbox"/> IsPaid	<input type="checkbox"/> ItemPrice	

SELECT statement:
SELECT DISTINCT [UserId], [UserName] FROM [UsersAndItemsView] ORDER BY [UserName]

On the Choose a Data Source page of the Data Source Configuration wizard, I set UserName as the value to display in the control, and set UserId as the value of the control, as shown in Figure 12-35. That means that the control will show user names. But, once the user makes a selection, the value of the control will actually be that user's UserId — the GUID that's automatically assigned to each new user account. We need the drop-down list's value to be that UserId, because that's the value that provides the link to transactions.

The Enable Postback option for the DropDownList control's Common Tasks menu must be selected for the page to work. Because that postback (combined with some filtering you'll see in a moment) is what allows the GridView control to show only transaction information for the user whose name is selected in the control.

Figure 12-35:
Data source
for the
**DropDown
List**
control in
Master-
Details.

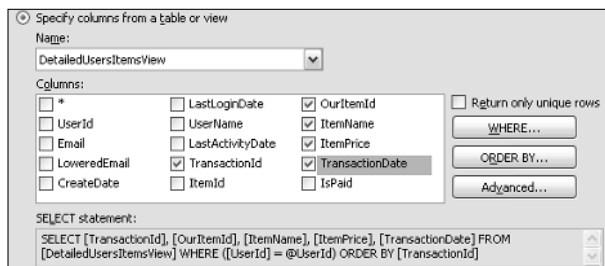


To customize the appearance of the `DropDownList` control, first click the control to select it. Then use its Properties sheet to refine the control's font and width.

Master-Details GridView control

The `GridView` control, named `GridView1` in the Master-Details example, is bound to the `DetailedUsersItemsView` view created back in Chapter 11. It uses a few fields from that view to make it easy for the user to locate a specific transaction, as shown in Figure 12-36.

Figure 12-36:
Selected
view and
columns
used for the
Master-
Details
**GridView
1** control.



The critical factor in getting the `GridView` control to show only records for the user whose name appears in the `DropDownList` control is in the `WHERE` clause. After clicking the `WHERE` button in Figure 12-36, I set up the condition as shown in Figure 12-37. Recall that the value of the `DropDownList1` control will be one person's `UserId`. Therefore, the condition specified in Figure 12-37 is for the `GridView` to show only records where the `UserId` field in the view matches the `UserId` that's in the `DropDownList1` control.

Figure 12-37:
The **WHERE**
condition
for Master-
Details
GridView
control.

Column:	UserId	Parameter properties
Operator:	=	Control ID:
Source:	Control	DropDownList1
SQL Expression:	[UserId] = @UserId	Default value:
		Value:
		DropDownList1.Selectedvalue

After finishing the Data Configuration Wizard for the `GridView` control, you need to choose `Enable Select` from its `Common Tasks` menu. Doing so places the `Select` option to the left of each record. As you'll see in a moment, the `DetailsView` control can then be configured to show only the record that's represented by the selected row in the `GridView` control.

The Master-Details `DetailsView` control

The `DetailsView` control in the Master-Details example needs to be filtered so that it displays only the transaction associated with whatever row is selected in the `GridView`. Also, that control needs to be bound to a single table that has a primary key. Otherwise, you couldn't use the control to edit or delete records.

In this example, the `Configure the Select Statement` page for the `DetailsView` control looks like Figure 12-38. Note that it's bound to the `Transactions` table, and shows all fields from that table (the `*` is short for "all columns").

Figure 12-38:
Table and
columns
bound to the
**Details
View**
control.

Specify columns from a table or view

Name: Transactions

Columns:

- *
- UserId
- ItemId
- TransactionDate
- IsPaid
- TransactionId

Return only unique rows

WHERE...
ORDER BY...
Advanced...

SELECT statement:
SELECT * FROM [Transactions] WHERE ([TransactionId] = @TransactionId)

To ensure that the `DetailsView` control shows only data associated with the record that's selected in the `GridView` control, click the `WHERE` button and choose options as shown in Figure 12-39. Those options, in English, say "where the `TransactionId` in the `Transactions` table matches the `TransactionId` in whatever row is currently selected in the `GridView`."

Figure 12-39:
The **WHERE**
condition
for the
Details
View1
control.

Column:	TransactionId	Parameter properties
Operator:	=	Control ID:
Source:	Control	Default value:
SQL Expression:	[TransactionId] = @TransactionId	Value:
		GridView1.SelectedValue

On the Common Tasks menu for the `DetailsView` control, choose `Enable Edit` and `Enable Delete` to display the `Edit` and `Delete` options that allow the user to edit the transaction.

When you first open a page like the `Master-Details` example, the `DetailsView` control will not be visible in the Web browser. That's because it's filtered to show only the transaction associated with the row that's selected in the `GridView`. So when there's nothing selected in the `GridView`, there's nothing for the `DetailsView` control to show. But as soon as you click `Select` at the left side of the `GridView`, your `DetailsView` control will appear showing the selected transaction record.

As an alternative to showing nothing when the `DetailsView` control has no record, you can edit its `EmptyData` Template to show a message. Here's how:

1. In `Design` view, open the `DetailsView` control's `Common Tasks` menu using the little triangle button or by right-clicking the control and choosing `Show Smart Tag`.
2. On the menu, click `Edit Templates`.
3. In the drop-down list that appears, choose `EmptyData` Template. A small box appears in which you can type and format text. For example, you might type `Select a transaction to see details`.
4. Right-click the template you just filled and choose `End Template Editing`.

From now on, when you first open the page in a Web browser, you'll see whatever text you typed into the `EmptyData` Template rather than nothing. As soon as you select a transaction from the `GridView` control, that text will be replaced by the actual `DetailsView` control showing the selected transaction.

General GridView and DetailsView considerations

The `GridView` and `DetailsView` controls are both new in ASP.NET 2.0, and offer easier, better editing capabilities than the older `DetailsView` and

FormView controls described later in this chapter. Both GridView and DetailsView have an Auto Format option on their Common Tasks menu, making it easy to define a general look and feel for the control right off the bat.

Both GridView and DetailsView also have an enormous number of properties you can tweak to get exactly the look and feel you want. There are so many properties that it would take more pages than there are in this book to describe them all. But rather than worry about every single property, a good general strategy might be to use Auto Format to get a general look for the control. Then use Properties to refine details, but only if you want to change something about the format.

Beyond all the properties for the control as a whole, there are many properties for each column in a GridView, and each field in a DetailsView. To get to those properties, you have to choose Edit Columns or Edit Fields from the control's Common Tasks menu. Then, under Selected Fields in the dialog box that opens, click on a specific field name under Selected Fields. The properties for that one column or field appear in a Properties sheet within the dialog box.

Remember that you can use both GridView and DetailsView either to show data or to allow editing of data in the database. But if you want to use either control to edit data, you must bind the control to a single table that has a primary key. Also, when using the DetailsView control to edit data, you need to remember to click the Advanced button in the Configure Select Statement page, as described under “Binding a DetailsView control” earlier in this chapter.

The DataList and FormView controls described later in this chapter are older ASP.NET controls. Although they hint at offering data-editing capabilities, they are much harder to use for that purpose. Your best bet is to try to use GridView or DetailsView when you know, in advance, that you want to edit data in the underlying database. Use DataList and FormView when you're just concerned with showing data on a page, without letting users make changes to that data.

Using the DataList Control

The DataList control is ideal for when you want to show data to users as though it were normal text, as opposed to items in a table or on a form. The DataList control's biggest strength is that it allows you to list data in just about any format imaginable. Using the DataList control is much like using the other Data controls you've seen in this chapter. Just follow these steps:

- 1. Create or open an .aspx page so you're in Design view.**
- 2. In the Toolbox, expand the Data category of controls.**

3. Drag a `DataList` control from the Toolbox onto the page (or Content Placeholder on a page that has a Master Page).
4. From the `DataList` control's Common Tasks menu, select Choose Data Source → <New data source...>.
5. As usual, choose Database and click OK.
6. Choose your usual connection string, and click Next.

That gets you to the standard Configure the Select Statement page.

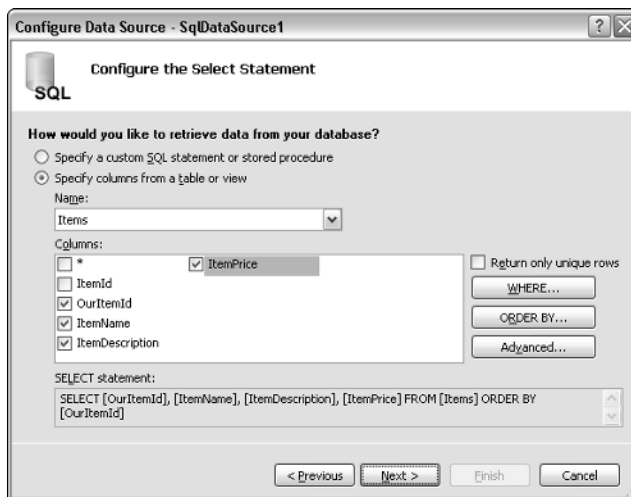
7. In the Configure the Select Statement page, choose the table or view from which the control will retrieve data.

As always, you can use the WHERE button to limit the records shown. Use the ORDER BY button to specify a sort order.

Figure 12-40 shows an example where I've opted to show all fields except `ItemID` from the `Items` table. I also used the ORDER BY button to set the sort order to `OurItemID`, Ascending order (though the only place you can see that is in the `ORDER BY` clause at the end of the `SELECT` statement).

8. Click Next and Finish to return to the page.

Figure 12-40:
Sample field
selections
for a
`DataList`
control.



In Design view, the bound `DataList` control appears as a placeholder of field names with “abc” placeholders for text, and random values like 0 and 0.1 as placeholders for numbers. When you view the page in a Web browser, you get a better idea of how the data will look to a person viewing the page through a Web browser. Figure 12-41 shows how the fields from the `Items` table look when viewed through an unmodified `DataList` control.

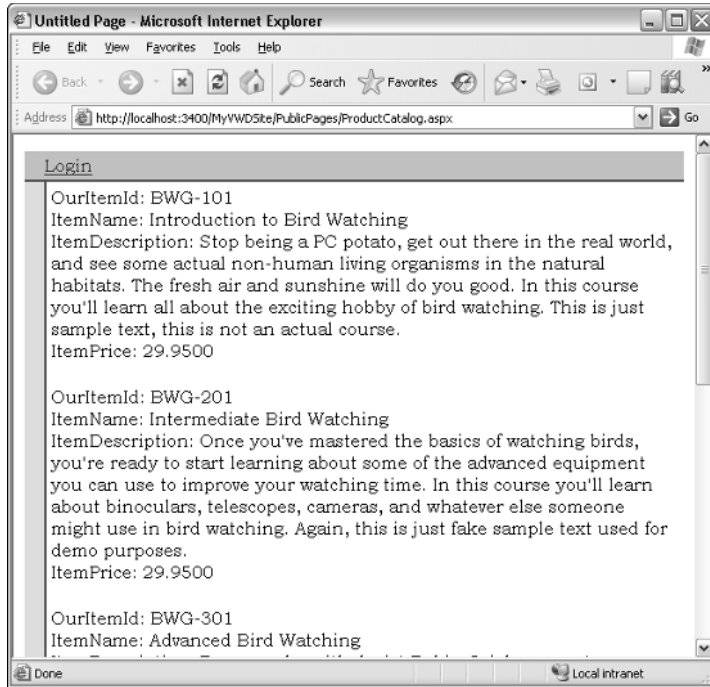


Figure 12-41:
Fields from
Figure 12-40
as displayed
by a
DataList
control.

Formatting a DataList control

The real beauty of the `DataList` control comes into play when you edit its template. To do so, click the `DataList` control to select it, then click the tiny triangle button near its upper-right corner. Or right-click the `DataList` control and choose Show Smart Tag. From the Common Tasks menu that appears, choose Edit Templates. The control takes on a completely different look, as shown in Figure 12-42.

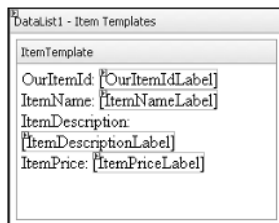


Figure 12-42:
A
DataList
control Item
Template.

The Item Template is like a small page that you can edit using standard word processing techniques and HTML. Text that's not enclosed in square brackets is just plain text typed right into the control. You can edit, format, or delete

that text however you like. For example, you can select a chunk of text and use options on the Formatting toolbar to make it bold, change its font, or whatever.

Items enclosed in square brackets are `Label` controls that are bound to data in the underlying database table or view. Those you can format in a similar manner. But you don't drag the mouse pointer through the item as you would with regular text. You just click the item to select it. Then choose options from the Formatting toolbar.



To select multiple `Label` controls, click one to select it. Then hold down the `Ctrl` key while clicking others that you want to select.

You can also rearrange things in the `ItemTemplate` to change how things look on the page. Moving things around can be a little awkward, especially if you're trying to align controls side-by-side. That's because the items word-wrap within the template, just as they would in a word processing program. To get things side-by-side, you may need to widen the control. To do so, click the top of the `DataList` control to select it. Then use the sizing handle on its right border to widen or narrow the control.

To move an item, just drag it to where you want to put it. Or, click the item to select it, press `Ctrl+X` to cut it, click where you want to put the control, and press `Ctrl+V` to paste it. If you want a space to appear between items in the page, make sure to insert a blank space between the items within the template.

Figure 12-43 shows an example where I deleted all the text labels, leaving only the bound `Label` controls. Those I rearranged so that the `ItemNameLabel` is on the top. The `ItemDescriptionLabel`, `OurItemIdLabel`, and `ItemPriceLabel` controls are side-by-side on one line, with one space between them.

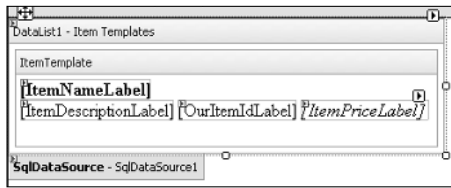
Blank lines in a `DataList` control

Though you can't see them, any extra carriage returns at the bottom of the control will add height to the control, and will also put extra space between each record on the Web page. (A carriage return is an invisible character that gets added every time you press the `Enter` or `Return` key.)

To reduce the height of a `DataList` control, you may need to remove some of those extra

carriage returns. You'll need to be in the `Edit Templates` mode, like Figure 12-42, to do that. Click inside the template, then move the cursor to the bottom of the template (press `Ctrl+End`). From there, each time you press `Backspace` you'll remove one carriage return (one blank line). If you go too far and delete a control within the template, press `Undo (Ctrl+Z)` to undo the deletion.

Figure 12-43:
DataList
 control's
 Item
 Template
 radically
 rearranged.



I also boldfaced the `ItemNameLabel` control and italicized the `ItemPriceLabel` control, just to illustrate the idea. I'll show you how that `DataList` control looks in a Web browser in a moment. First, let's talk about . . .

Formatting dates and numbers in a DataList

To apply a date or currency format to a field, click its `Label` control to select the control. Then, click its tiny triangle button to show its `Common Tasks` menu as shown in Figure 12-44. Then, choose `Edit DataBindings`.

Figure 12-44:
 Common
 Tasks for
 a single
 bound
Label
 control.



A `DataBindings` dialog box for the selected control opens. If the item you're formatting is a currency or date value, use the `Format` drop-down list to choose a format for the item. For example, Figure 12-45 shows the `DataBindings` dialog box for the `ItemPrice` control in my sample `DataList`. You can see where I've chosen `Currency - {0:C}` from the `Format` drop-down list to display that value in `Currency` format.

Figure 12-46 shows the results in a Web browser. The arrangement and appearance of data on the page is a direct reflection of the arrangement of the `Label` controls shown back in Figure 12-43. Each `ItemPrice` is shown in `Currency` format (\$29.95) rather than the default 29.9500 format.

Figure 12-45:
The Data Bindings dialog box for the `ItemPrice` Label control.

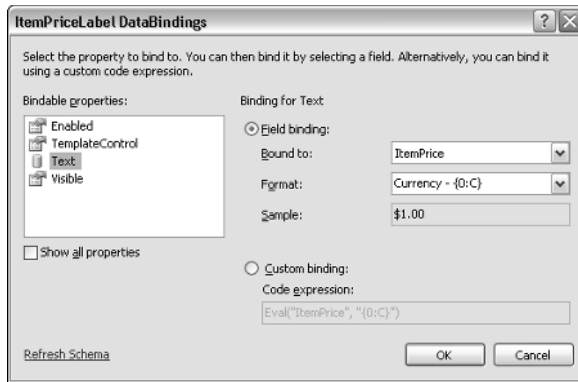
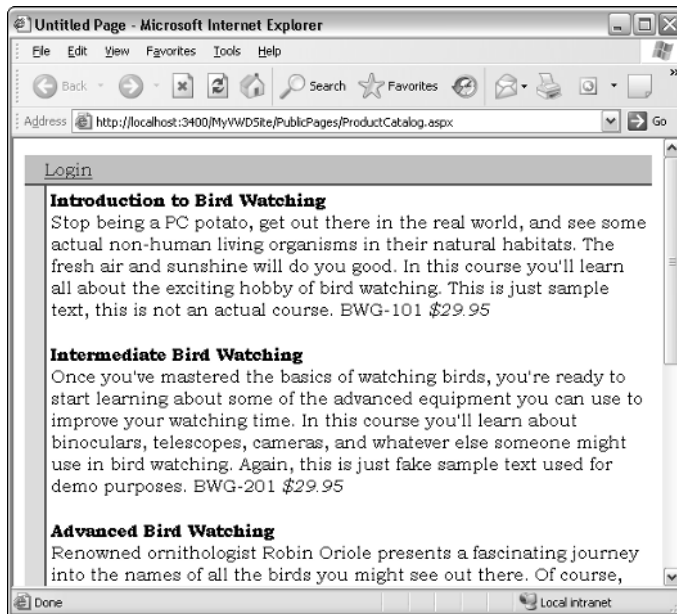


Figure 12-46:
The `DataList` control from Figure 12-43 in a Web browser.



Showing a `DataList` in columns

The `DataList` control can also show data in newspaper-style columns. To use this feature, you'll need to end Template Editing if you're still viewing the Item Template in the Design surface. Right-click the `DataList` control and choose End Template Editing, or choose End Template Editing from the `DataList` control's Common Tasks menu. Then, click the tiny triangle near the upper-right corner of the `DataList` control (not a single `Label` control) and choose Property Builder.

In the Properties dialog box that opens, click on General in the left pane. Then specify the number of columns you want to display from the Columns option.



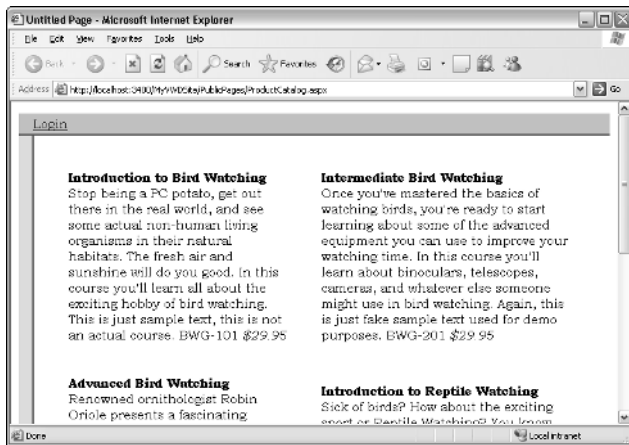
For the Layout option, you need to choose Table. If you choose Flow, the columns won't work. Use the Flow option only when you want data displayed without a table.

Use the Direction option to choose a Horizontal or Vertical orientation for the columns. If you choose Horizontal, the first record's data appears first, the second record's data appears in the column to the right, and so forth. If you choose Vertical, the first record's data appears first, the second record's data appears under that one in the same column, then the third record, and so forth down the page.

To put some space between columns, click the Borders option at the left side of the Properties dialog box. Then set the Cell Spacing setting to the number of pixels to put between each column. Optionally, if you want to show border lines on the page, choose an option from the Grid Lines drop-down list. Then choose a color and width for the borders.

To see the results of your changes, click OK to close the Properties dialog box. Then view the page in a Web browser. Figure 12-47 shows an example where I set Columns to 2, Direction to Horizontal, and Cell Spacing to 40. It's the same data as in Figure 12-46 split into two columns.

Figure 12-47:
The
DataList
control
showing
data in two
columns.



Using DataList to show pictures

In Chapter 11, I created a sample table named Photos that contains two text fields named PhotoCaption and PhotoURL. The PhotoCaption field contains

plain text, while the `PhotoURL` field contains the path to a picture in a folder named `FlowerPix`. You can use a `DataList` control to display the actual picture to which each `PhotoURL` refers.

There's nothing special about how you initially create the control. The steps, as usual, are:

1. Drag a `DataList` control from the `Data` group in the `Toolbox` onto any `.aspx` page or `Content Placeholder`.
2. From the `DataList` control's `Common Tasks` menu, select `Choose Data Source` → `<New Data Source>`.
3. Choose `Database`, then click `OK`.
4. Choose your standard connection string and click `Next`.
5. Choose your table or view and columns to display.

For this example, I'll retrieve the `PhotoCaption` and `PhotoURL` fields from the `Photos` table, as in Figure 12-48.

6. Click `Next` and `Finish`.

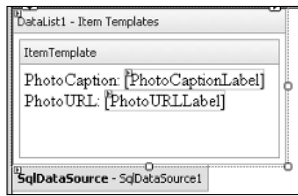
Figure 12-48:
Binding
`DataList`
control to
fields from
the `Photos`
table.



In `Design` view, the `DataList` control appears with the usual field names and placeholders (abc for the text fields). In a `Web` browser, you still only see the links to the images, for example `~/Images/FlowerPix/Flower01.jpg` rather than the actual photo. To get the photo to show, you need to edit the `DataList` control's `ItemTemplate`.

As always, you can edit the template by choosing `Edit Templates` from the control's `Common Tasks` menu, or by right-clicking the control and choosing `Edit Templates` → `Item Template`. The `ItemTemplate` is like a miniature page that defines how each record from the underlying table will be displayed on the page. Initially the `ItemTemplate` just shows each field name as text typed right into the template, and each field as a `Label` control, as shown in Figure 12-49.

Figure 12-49:
The
ItemTemplate
for a
DataList
control.



To show each picture as an actual picture rather than the `PhotoURL` value, you need to get an `Image` control from the Standard group of tools into the `ItemTemplate`. Then bind the `ImageUrl` property of that control to the `PhotoURL` field. You can start by deleting what's already in the template. Here are the steps:

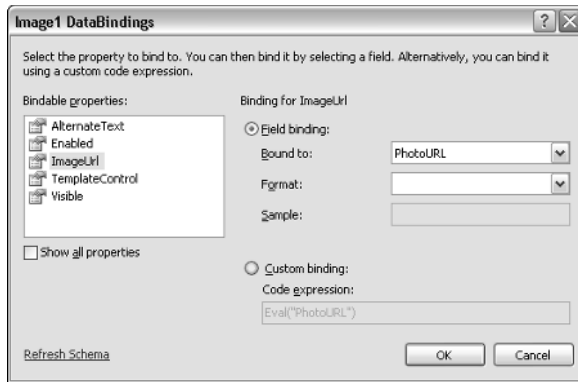
1. Delete all the text and both `Label` controls from the `ItemTemplate`, so that the template is just an empty box.
2. In the Toolbox, click the + sign next to Standard to show the Standard ASP.NET controls.
3. Drag an `Image` control from the Toolbox into the `ItemTemplate`.
The control appears as a small box containing a red X (it's just a placeholder for each picture that will be shown in the Web browser).
4. Click the `Image` control's Common Tasks button and choose Edit Data Bindings.
5. In the dialog box that opens, click `ImageUrl` in the left column under Bindable Properties.
6. In the right column, choose Field Binding.
7. From the Bound To drop-down list, select the name of the field that contains the link to the image.

In the example of the `Photos` table, that's the `PhotoURL` field, as shown in Figure 12-50.

8. Click OK.

Viewing the page in a Web browser at this point will display each picture rather than just the URL to the picture. If that's all you want, then you're done. But chances are you'll want something fancier than just a bunch of pictures shown down the page. So let's look at some things you can do to fancy things up.

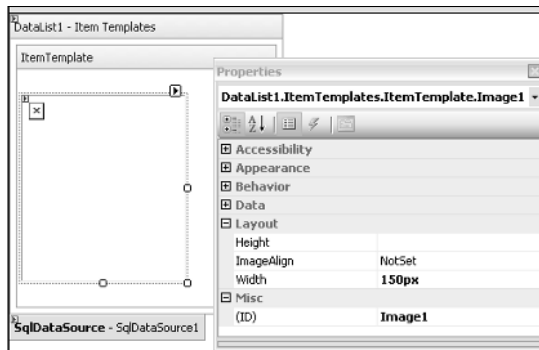
Figure 12-50:
The
ItemTemplate
for a
DataList
control.



Sizing the pictures

You might want to make each picture of equal size so you can put them into columns. Click on the `Image` control to select it. Then, in the Properties sheet, set the `Width` property to however wide you want each picture to be. For example, in Figure 12-51, I've set the `Width` of the `Image` control to 150 pixels.

Figure 12-51:
Width of the
`Image`
control set
to 150
pixels.



Showing pictures in columns

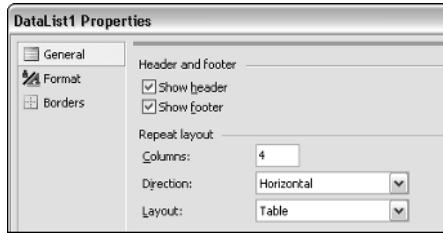
Recall that the `DataList` control can show data in multiple columns. To show pictures in columns, you first need to get out of Template Editing mode. Then use the Property Builder for the `DataList` control to configure the columns. The steps are:

1. **Right-click the `ItemTemplate` and choose `End Template Editing` (or choose `End Template Editing` from the `Common Tasks` menu).**
2. **Choose `Property Builder` from the `DataList` control's `Common Tasks` menu.**

3. In the Properties dialog box, set the Columns option to the number of columns you want, then choose either Horizontal or Vertical for the orientation.

For my example, I set Columns to 4 and Direction to Horizontal, as shown in Figure 12-52.

Figure 12-52:
Going
for four
columns of
photos here.



4. Optionally, choose other formatting options from the Properties dialog box. For the example you're about to see, I did the following:

- Click Format in the Properties dialog box, click the + sign next to Items, click Normal Items, set the Horizontal Alignment to Center, and set the Vertical Alignment to Bottom.
- Click Borders in the Properties dialog box and set the cell spacing, padding, and border lines. For this example, I only set Cell Spacing to 10 to put a little space between each picture.

5. Click OK in the Properties dialog box.

Viewing the page in a Web browser at this point shows the pictures in columns, as in the example shown in Figure 12-53.



Figure 12-53:
Pictures
displayed
in columns
by a
DataList
control.



Adding picture captions

Let's suppose that you also want to show each picture's caption. But as an added bonus, you want to make each caption a link that, when clicked, shows the corresponding image at full size, rather than with the Width you set for the Image control. The trick here is to add a HyperLink control to the template. Set the Text of that link to the PictureCaption field, and the NavigateURL property to the PictureURL field. Here are the steps:

1. In Design view, choose **Edit Templates from the DataList control's Common Tasks menu**.
2. In the ItemTemplate, click to the right of the Image control and press Enter to make a little room under the control.
3. From the Standard group of tools in the Toolbox, drag a HyperLink control into the ItemTemplate, so that it's under the Image control.
4. From the HyperLink control's Common Tasks menu, choose **Edit DataBindings**.
5. Set the Text property to the name of the field that contains the text to show.

In my example, that would be the PhotoCaption field, as shown in Figure 12-54.

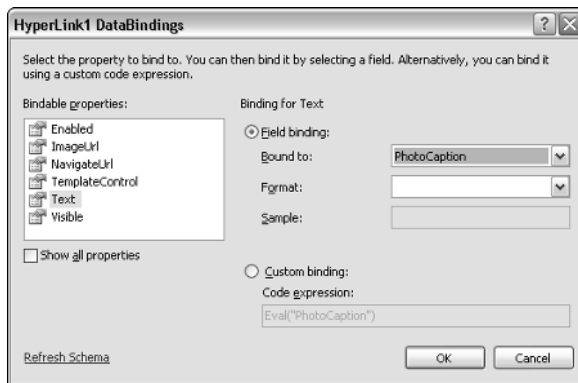
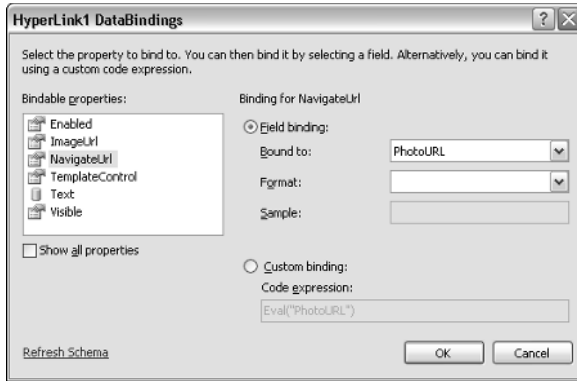


Figure 12-54:
The Text property for the sample HyperLink control.

6. In the left column, click on NavigateURL to define where the link will take the user.
7. Set the Bound To property for the NavigateURL property to the name of the control that contains the path to a page or picture.

In this example, that would be the PhotoURL field, as shown in Figure 12-55.

Figure 12-55:
The
**Navigate
URL**
property for
the sample
**Hyper
Link**
control.



8. Click OK in the dialog box.

Figure 12-56 shows how the page looks when viewed in a browser. When a user clicks a picture caption, the effect will be to open the picture, full size, in a new browser window. The user can then click the Back button in the browser to return to the previous page where all pictures are shown in columns.



Figure 12-56:
Photos with
captions
displayed as
hyperlinks.



To download a picture, the user just needs to right-click the picture and choose Save Picture As. You could add instructions to the page to explain that to the user.

Using a DataList to show HyperLinks

In the photos example, I used a `HyperLink` control in a `DataList` `ItemTemplate` to show hyperlink text from a table (the `PhotoCaption` field), and to define where the link takes the user (the `PhotoURL` field). You can use the same technique to show a list of links that are stored in a Database table.

In Chapter 11, I created a sample table named `Links` that contains two text fields named `SiteName` (the name of a Web site) and `SiteURL` (the URL of the site). It's a small sample table, as shown in Figure 12-57. But of course the technique described here would work with any number of records in a table.

Figure 12-57:
Sample
Links table
with site
names and
URLs.

Links: Query(...ASPNETDB.MDF)		
SiteId	SiteName	SiteURL
1	ASP.NET	http://asp.net
2	Microsoft Developer Network	http://msdn.microsoft.com
3	DotNetJunkies	http://www.dotnetjunkies.com/
4	.netECommerce	http://www.dotnetecommerce.com/
5	XHTML 2.0 Spec	http://www.w3.org/MarkUp/
6	CSS Home Page	http://www.w3.org/Style/CSS/
*	NULL	NULL

A `DataList` control containing a standard `HyperLink` control that's bound to the `SiteName` and `SiteURL` fields in the table will present a nice list of links. Here are the steps to create the list:

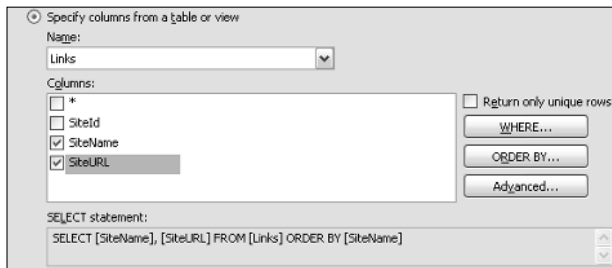
1. From the **Data** group in the **Toolbox**, drag a **Data List** control to any **.aspx** page or content placeholder.
2. From the **DataList** control's **Common Tasks** menu, select **Choose Data Source** → **<New Data Source>**.
3. Choose **Database** and click **OK**.
4. Choose your normal connection string and click **Next**.
5. Choose the table that contains the site data, then choose the columns that contain the site name and URL.

For my example, that would be the `SiteName` and `SiteURL` fields in the `Links` table, as shown in Figure 12-58.

6. Optionally, use the **ORDER BY** button to alphabetize the links by **SiteName**.
7. Click **Next**, then click **Finish**.

Figure 12-58:

Binding control to the SiteName and SiteURL fields in the Links table.



You see the control with the standard field names and `abc` placeholders. To change that to a list of working hyperlinks, you need to remove the text and labels that are currently in the control. Then put in a single `HyperLink` control. Here's how:

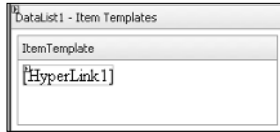
1. Choose **Edit Templates** from the `DataList` control's **Common Tasks** menu.
2. Delete the text and `Label` controls that are currently in the template, so that the template is empty.
3. From the **Standard** group of controls in the **Toolbox**, drag a `HyperLink` control into the **Template**.
4. From the `HyperLink`'s **Common Tasks** menu, choose **Edit DataBindings**.
5. In the left column of the dialog box that opens, click **Text**, then set the **Bound To** property to the name of the field that contains the site name.
In my example, that would be the `SiteName` field.
6. In the left column, click the **NavigateURL** property.
7. Set the **Bound To** property to the name of the field that contains the URL.
In my example, that would be the `SiteURL` field.
8. Click **OK**.

At this point, the `ItemTemplate` looks like Figure 12-59.

When viewed in a Web browser, the page shows each site's name as a hyperlink. Clicking the link opens the page to which the link refers.

Figure 12-59:

DataList control's ItemTemplate contains a HyperLink control.

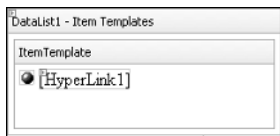


Keep in mind that the ItemTemplate is like a mini Web page onto which you can place any text or image. For example, let's say you have a small graphic image of a bullet you'd like to place to the left of each list item.

First, you'll need to make a little space to the left of the HyperLink control. Click inside the template, press Ctrl+Home a couple of times, then press the spacebar and left arrow key to make a little space. Then, just drag your little image into the space you made, as in Figure 12-60.

Figure 12-60:

Graphic in ItemTemplate (left) and in browser (right).



- [.netECommerce](#)
- [ASP.NET](#)
- [CSS Home Page](#)
- [DotNetJunkies](#)
- [Microsoft Developer Network](#)
- [XHTML 2.0 Spec](#)

If you want to put a bullet character and space to the left of each item, rather than a graphic image, switch to Source view by clicking the Source button at the bottom of the Design surface. Find the `<asp:HyperLink...>` tag between the `<ItemTemplate>` and `</ItemTemplate>` tags. To the left of that `<asp:hyperlink...>` tag, type **•** (for the bullet) and ** ** (non-breaking space) for the blank space. Both characters should be right up against the `<asp:Hyperlink...>` tag like this:

```
&bull;&nbsp;<asp:HyperLink ID="HyperLink1"...>
```

When you switch back to Design view, by clicking Design at the bottom of the Design surface, you'll see the character and space in place. Of course, in the Web browser, the character will appear to the left of each item in the list.

As you've seen, the `DataList` control offers lots of different ways to display data from database tables. The template you define is applied to every record that's retrieve from the database table. But sometimes, you don't want to show data from multiple records. Sometimes, you just want to show a single record, or a single piece of information from a table. For those occasions, the `FormView` control might be your best bet.

The FormView Control

The `FormView` control is similar to the `DetailsView` control in that it only shows one record at a time. But it's also similar to a `DataList` control in that it displays data in such a way that it looks like text printed on a page rather than a table or form.



If you want to display and edit one record at a time, you're much better off using the `DetailsView` control, as opposed to `FormView`. The `DetailsView` control, which is new in ASP.NET 2.0, is much easier to work with.

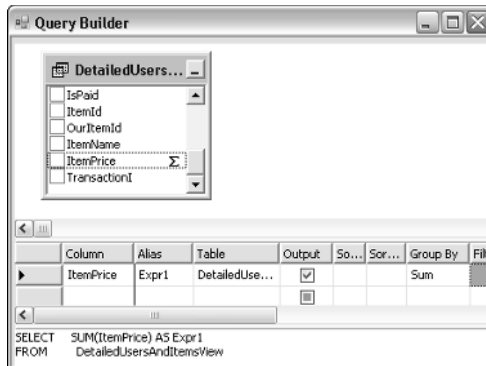
The `FormView` control works like the other Data controls in that you drag it to a page, define your data source, and optionally edit its template to get the exact look you want. Because it's limited to displaying a single record, `FormView` can be useful for displaying a single item of data, such as a sum or total. Here's an example.

In Chapter 11, I created a view named `DetailedUsersItemsView` which contains detailed information about every transaction. Suppose you want to sum up the `ItemPrice` field in that view to see a grand total of all transactions. You'd start by dragging a `FormView` control onto a page. From its Common Tasks menu, choose `New Data Source`, choose `Database`, and click `OK`. As always, choose the standard connection string, and click `Next`.

On the `Configure the Select Statement` page, you won't be able to use the "Specify columns from a table or view" in this example because you need a sum of all the numbers in a view. You have to choose "Specify a custom SQL statement or stored procedure," then click `Next` to get to the more advanced query editor where you can specify a sum. On the next page that appears, click the `Query Builder` button.

In the `Add Table` dialog box that opens, click the `Views` tab, click `Detailed UsersItemsView`, click `Add`, then click the `Close` button in the `Add Table` dialog box. In the `Query Builder`, scroll down the field list for the view and choose `ItemPrice`. Right-click some empty space in the top page of the `Query Builder` and choose `Add Group By`. Then, in the query grid, set the `Group By` column for the `ItemPrice` field to `Sum`, as shown in Figure 12-61.

Figure 12-61:
A query to
sum the
ItemPrice
field in
Detailed
Users
ItemsView.



Click OK at the bottom of the Query Builder. Then click Next, and then click Finish. In Design view, you end up with a `FormView` control that just shows:

```
Expr1: 0
```

Viewing that page in a Web browser shows something like the example below (though the number will be the sum of whatever values are in the `ItemPrice` field of the view at the moment):

```
Expr1: 469.4000
```

As it stands, the information presented by the control isn't exactly self-explanatory. But you can edit the control's template to show anything you want. First you need to close the Web browser (if it's open) to get back to the Design view of the page. Then right-click the `FormView` control, choose Show Smart Tag, then choose Edit Templates.

To show the sum in currency format rather than in the 469.4000 format, do as you did in the `DataList` control. That is, right-click [Expr1Label], choose Show Smart Tag, then choose Edit Data Bindings. Set the Format to `Currency - {0:C}` and clear the "Two-way databinding" check box as shown in Figure 12-62, and click OK.



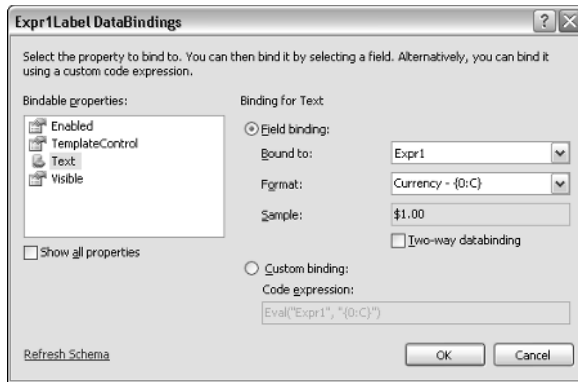
You need only choose "Two-way databinding" if you intend to edit data in a `FormView` control. However, if you want to edit data, you'd be better off using a `DetailsView` control, as it's a lot easier with `DetailsView`.

In the `ItemTemplate`, replace the text `Expr1:` with `Grand Total:` as shown in Figure 12-63.

When you view the page in a browser, the control shows:

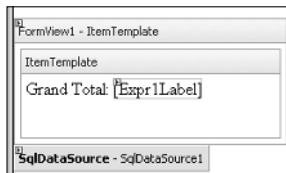
```
Grand Total: $469.40
```

Figure 12-62:
Data-
bindings for
the Expr1
Label
control.



which is the sum of all the `ItemPrice` fields in `DetailedItemsUsersView`, expressed in a more meaningful format than `Expr1`: 469.4000.

Figure 12-63:
Item-
Template for
sample
FormView
control.



Showing subtotals

Showing just the grand total of sales on a Web page is useful, but chances are you might also want to see things subtotaled by customer, product, or month. You could add a `GridView` control on the same page as the Grand Total, and have that `GridView` control show multiple records with the appropriate grouping.

For example, with the previous `FormView1` control still on the page, press `Ctrl+Home` a couple of times, and press `Enter` a couple of times, to make some space above the `FormView` control. Then drag a `GridView` control into that space at the top of the page.

From the `GridView`'s `Common Tasks` menu, select `Choose Data Source` > `<New Data Source>`, as usual. Then choose `Database` and click `OK`. Choose your usual `Connection String` and click `Next`. You'll come to the `Configure the Select Statement` page where you'll once again need to choose "Specify a custom SQL statement or stored procedure," then click `Next`.

In the next box that opens, click the Query Builder button to get to the more advanced query builder. In the Add Table dialog box that opens, click the Views tab, click `DetailedUsersItemsView`, click Add, then click Close to close the Add Tables dialog box.

How you set up the query depends on how you want to group the subtotals. First, right-click some empty space at the top of the Query Builder and choose Add Group By to add a Group By column to the grid. Then, if you want to subtotal by users, choose the `UserName`, `Email`, and `ItemPrice` columns from the Field List. Then set the Group By column to Sum for the `ItemPrice` field, as in Figure 12-64.

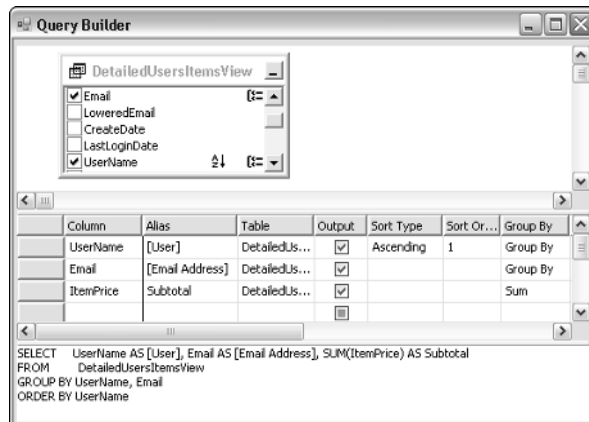


Figure 12-64:
Group By
and Sum
to subtotal
by User.

Notice in Figure 12-64 I also set the Alias column for each field. Whatever you type into the Alias column will appear at the top of the column in the GridView control. The default aliases will be generic (`Expr1`, `Expr2`, and so forth), which isn't particularly meaningful information to someone viewing a page. So you wouldn't necessarily want those names appearing at the top of the GridView control. The easy way to put more meaningful names at the top of a GridView control is to just change those generic names to whatever names you want to put at the top of the GridView.

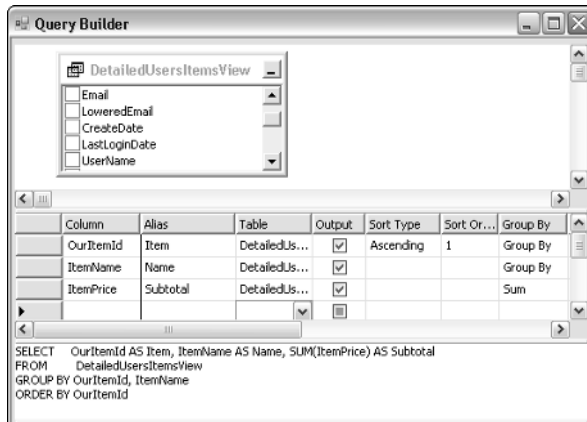
The square brackets you see in the figure around the `[User]` and `[Email Address]` names are only required when the column name might conflict with some other name. But you don't have to worry about that. After you type an alias name of your own choosing, the Query Builder will automatically add square brackets if (and only if) they're needed.



You can click the Execute Query button near the bottom of the Query Builder window to test the query. The query results appear in the bottom pane of the window.

The above example assumes you want to see subtotals based on users. Suppose that you wanted to subtotal by Item rather than by Customer. You could use the `OurItemId`, `ItemName`, and `ItemPrice` columns, with only the `ItemPrice` column set to `Sum`. Then enter meaningful aliases for the `GridView`'s column headings, as shown in Figure 12-65.

Figure 12-65:
Group By
and Sum to
subtotal by
Item.

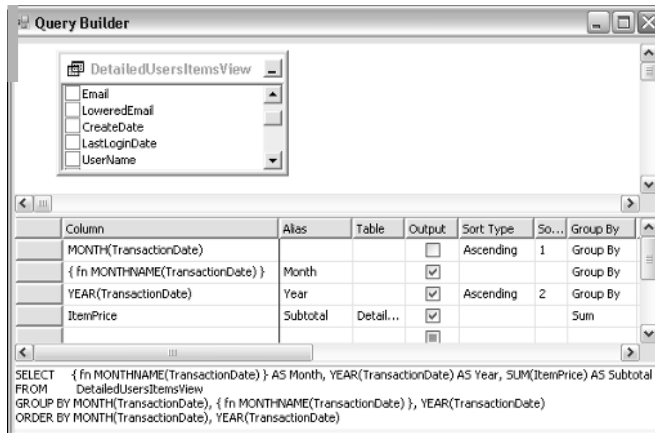


If you prefer to subtotal my month, rather than by customer or item, you'll need to use some custom expressions. In my example, the `TransactionDate` field contains the date of each transaction. You can use the following expressions to group by month number, month name, and year:

- ✓ **MONTH(TransactionDate):** Isolates the month number of the `TransactionDate` field. This field is for sorting purposes only, so you can clear its `Output` check box to hide it in the query results. Set its `Sort Type` to `Ascending`.
- ✓ **MONTHNAME(TransactionDate):** Isolates the month name from the `TransactionDate`. The Query Builder will change the syntax of this one slightly to read `{fn MONTHNAM(TransactionDate)}`. But don't worry about that. It happens automatically if, and only if, the specific expression you entered requires the modified syntax. You can just let it happen and not fret over it.
- ✓ **YEAR(TransactionDate):** Isolates the year of the `TransactionDate`. Set this item's `Sort Type` column to `Ascending` as well, so items are listed in month and year order.

Figure 12-66 shows how you'd use the expressions in the grid portion of the query. Notice that I'm sorting by the month number and year, to get the records to be displayed in chronological order. As always, all fields except `ItemPrice` are set to `Group By`, since only `ItemPrice` needs to be totaled.

Figure 12-66:
Group By
and Sum to
subtotal by
Month.



When you've finished defining the query, click OK in the Query Builder. Then click Next and Finish to return to the page. The only thing you'd really need to format in the `GridView` control would be the `ItemPrice` field, which will show in that 29.9500 format if you don't set its `Format` to `Currency`.

To format the `ItemPrice` column, choose `Edit Columns` from the `GridView`'s `Common Tasks` menu. Choose `Subtotal` from the `Selected Fields` box near the bottom of the dialog box. Then set that field's `DataFormatString` property to `{0:C}` as shown in the examples presented earlier in this chapter. While you're at it, click the `+` sign next to `ItemStyle` in the `Properties` list, then set the `HorizontalAlign` property to `Right` to have number right-align within the column. Then click OK.

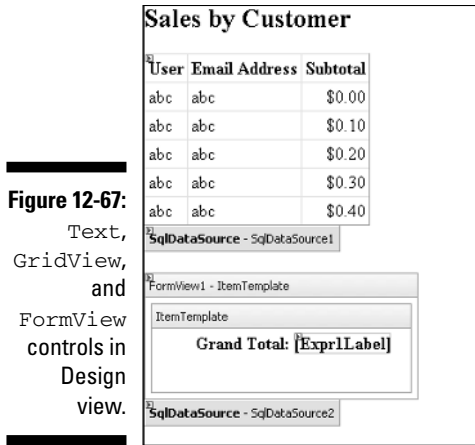
In Design view, the page won't look like much, just the usual placeholders. Figure 12-67 shows an example. The `Sales by Customer` text at the top of the page is just text I typed right into the page. The `GridView` is bound to the query shown back in Figure 12-64, which subtotals by user. The `FormView` control is the same as described at the start of this section.

When viewed in a Web browser, though, the page shows total sales by user, with a grand total at the bottom, as shown in Figure 12-68.



You could create three separate pages, one for each type of query, making it easy to view subtotals and totals by user, product, or month.

It's important to understand that all of the examples you've seen in this chapter are just examples. There's almost no limit to the things you can do with the `GridView`, `DetailsView`, `DataList`, and `FormView` data controls in Visual Web Developer. In this chapter, we've barely scratched the tip of the proverbial surface. But this should be enough to get you in the ball game, and to see how you can easily bind data controls to SQL server tables and views to extract whatever data you need to show on a Web page.



How easy, or how difficult, it all seems really depends on your perspective. Clearly, if you're already familiar with database concepts and SQL, things will seem easy because you can do so much without writing any code. If you're new to database management, you may want to invest some time boning up on database design basics and SQL to understand the full creative potential these controls offer.



Part IV

The Part of Tens

The 5th Wave

By Rich Tennant



"What you want to do is balance the image of the pick-up truck sittin' behind your home page, with a busted washing machine in the foreground."

In this part . . .

Welcome to the Part of Tens. Here you discover the actual plain-English meaning of the top ten buzzwords Web developers use to describe their tools and techniques. After that, you get a treasure map to ten online places where Web developers get their information, ask questions, and get answers.

Chapter 13

Ten Terms to Make You Look Smart

Even if this book were 3,000 pages long, it still wouldn't be long enough to cover everything there is to know about Visual Web Developer, HTML, CSS, ASP.NET, and programming. You'll often need to refer to online resources or books for more information.

Given that Visual Web Developer is a tool for software developers, you'll have to get accustomed to terminology that developers use. This chapter presents the top ten terms that everyone will assume you already know and understand.

Web Application

Familiar computer terms can change when they venture onto the World Wide Web. For instance, you may already know *application* as a program such as a word processor or spreadsheet, but the term *Web application* basically means a Web site or Web page that contains more than just static, unchanging HTML and text that looks the same to all visitors. A Web application is more dynamic than that; the page's exact content is created just before the page is sent. Exactly what the visitor sees depends on who that visitor is, or the specific information the user requested.

Developer

There are two main types of computer users in the world, those who *use* software, and those who *create* software. The people who create software are called *developers*. Everyone else is called a *user*.

It's a lot easier to be a user than a developer. As a user, you can often figure out how to do things just by guessing and hacking away until you get it right. Not so for a developer. You have to actually know how the technology works — as well as what you're doing with it — to create software. Guessing and hacking away at random generally leads to nothing but endless hours of hair-pulling frustration.

Data-Driven

A *data-driven* Web site consists of Web pages whose content depends upon (“is driven by”) the data stored in a database. For example, when you search the Internet with a search engine such as Google, you’re basically sending a query string to Google’s database. What you get back is a dynamic page created in response to your query; exactly what appears on it depends on what you searched for. The page’s specific content is “driven” (determined) by what’s in Google’s database that matches what you searched for.

ASP.NET 2.0

ASP stands for Active Server Pages, and is Microsoft’s fundamental technology for building dynamic, data-driven Web sites. As with all technologies, ASP has evolved over the years to become ever more powerful and easy to use.

As I write this book, ASP.NET 2.0 is the latest version of ASP. It’s also the version of ASP used by Visual Web Developer to create dynamic data-driven Web sites. To publish a site you created with Visual Web Developer, you must find a hosting provider that supports ASP.NET 2.0.

Visual Studio

Visual Studio is Microsoft’s main software-development tool. It has all kinds of functionality for creating all kinds of programs. It’s big, complex, and easy to get lost in because Visual Studio provides a seemingly-endless array of tools and options.

Even so, Visual Web Developer is (believe it or not) a greatly *uncomplicated* version of Visual Studio; it focuses on developing one type of application: dynamic, data-driven Web sites. In effect, Visual Web Developer just “hides” the parts of Visual Studio that aren’t relevant to developing Web applications.

IDE

IDE acronym stands for Integrated Development Environment. Both Visual Studio and Visual Web Developer are IDEs. They’re called *integrated* because they give you access to all the various tools and technologies you need to get the job done.

To create data-driven Web sites, for example, you need a pretty specific list of things: HTML, CSS, ASP.NET, a data source (such as SQL Server), and at least one programming language (such as C#). An IDE like Visual Web developer brings all those technologies together under one roof, so to speak, so you can get all your work done in one program window. (What a concept.)



You've seen a similar (but unrelated) term before: In hard drives, IDE stands for Integrated Device Electronics. But that concept is in no way related to software development. In hard drives, it just means that all the electronics the disk needs to operate are contained within the drive's case.

Control

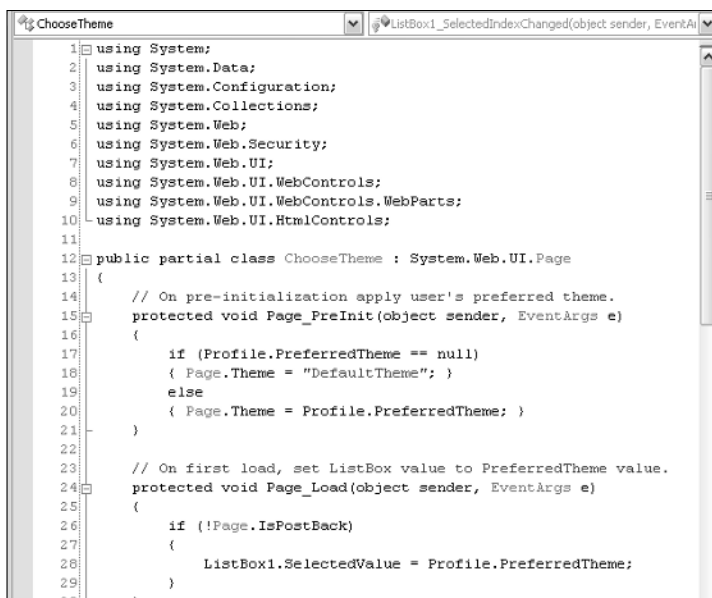
When used as a noun within the context of programming, the term *control* refers to things in a program, or on a page, that allow users to control the action. For example, open any Windows dialog box and you see a bunch of text boxes, drop-down menus, check boxes, and buttons. All of these are relatively small, simple controls.

In a more general sense, a control can be anything on the page that a user interacts with. In Visual Web Developer's Toolbox, each item actually represents a control that you can drag and drop onto a page. As a developer, you choose which controls users will see, and can program what happens when a user interacts with a control.

Code

Code is the text that forms the instructions for the computer, written in some programming language such as C# or Visual Basic. It has a different function from text that's typed in a human language. Unlike regular text, code uses no sentences or paragraphs. Instead, code consists of *statements* (lines) written in the programming language rather than in a human language. Figure 13-1 shows an example of some C# code.

When you write in (say) English, you can get away with murder in terms of spelling and punctuation; even if the text would make your composition teacher cry, your human recipient will probably still be able to figure out what you mean — simply by studying the context of the message. (If you've spent any time in online forums, you've no doubt seen the extent to which people can get away with worse-than-iffy spelling and punctuation.)



```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Collections;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.WebControls;
9 using System.Web.UI.WebControls.WebParts;
10 using System.Web.UI.HtmlControls;
11
12 public partial class ChooseTheme : System.Web.UI.Page
13 {
14     // On pre-initialization apply user's preferred theme.
15     protected void Page_PreInit(object sender, EventArgs e)
16     {
17         if (Profile.PreferredTheme == null)
18         { Page.Theme = "DefaultTheme"; }
19         else
20         { Page.Theme = Profile.PreferredTheme; }
21     }
22
23     // On first load, set ListBox value to PreferredTheme value.
24     protected void Page_Load(object sender, EventArgs e)
25     {
26         if (!Page.IsPostBack)
27         {
28             ListBox1.SelectedValue = Profile.PreferredTheme;
29         }
30     }
31 }
```

Figure 13-1:
Some
sample
code written
in C#.

When you write code, there is no such margin for error. Every character of every word counts; if it isn't just so, you'll likely get an error message rather than code that works. Unlike a human being who can figure out (at least) the intent of some dreadfully composed written message, a computer can't "figure out" diddley squat. You either write the code correctly and it works, or you write it incorrectly and it doesn't work.

Programmatic

There are two ways to interact with controls. One way is manual: You actually *use* the finished control yourself. For example, when you click a button in a Windows dialog box, you are manually operating the control with your mouse and keyboard.

As a programmer, you can also write code to specify the exact appearance and behavior of a control, as well as what happens after a user interacts with the control. That sort of code is generally executed in response to some event, as when the user clicks a button or makes a selection from a drop-down menu.

In essence, a user who is viewing a page and makes a selection from the options provided *is* manually using the control. But exactly what happens after the user makes a selection is generally handled *programmatically* — the control runs some code you’ve written, and that code tells the machine exactly what to do.

Database

The term *database* is often used casually to refer to any body of knowledge or collection of information. In programming terms, it’s more specific — referring to data that’s organized into tables that consist of columns and rows so it can be stored and accessed consistently. The database for a Web site is made available through the Database Explorer tab in Visual Web Developer, as shown in Figure 13-2.

A database management system (DBMS) is a program that allows you to organize data into tables, and retrieve data programmatically. Microsoft Access and SQL Server are examples of database-management systems. Microsoft Access is a self-contained DBMS; it contains all the tools you need to create tables, queries, forms, reports, and other aspects of an application — in one program.

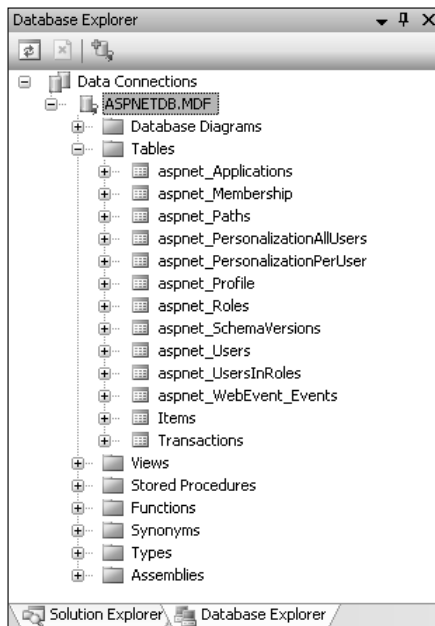


Figure 13-2:
A Web site's
database.

SQL Server isn't so much an application as it is a *server* — a single-minded program obsessed with serving up data. Unlike an application program, which has its own program window, a server works behind the scenes; it has no particular interface or program window of its own. Instead, the server just “serves up” data to some external application — such as (for a convenient example) a dynamic data-driven Web site you create in Visual Web Developer.

Chapter 14

Ten Alternatives to Being Helpless

Visual Web Developer brings together all the technologies you need to create dynamic, data-driven Web sites. Those technologies include HTML, CSS, ASP.NET, C#, the .NET Framework, and SQL Server — each and every one of which is an enormous topic in itself. (For example, the first printed documentation for the .NET Framework alone consisted of over 8,000 pages, bound into several volumes.)

To deal with such a gigantic body of knowledge, you must find ways to get exactly the information you need, when you need it. Sending desperate “PLEASE HELP!!!!” messages generally won’t cut it. You have to be more resourceful — and know where the people you’re asking are getting their information, so you can go get that information too.

The sites listed in this chapter get you in the ballpark. Of course, Web sites being what they are, some of the URLs listed here may change between now and the time you actually read this. That’s why I keep an up-to-date list of links on www.coolnerds.com/vwd just in case any change occurs. If you have any problems finding a page, try looking there.

Microsoft Developer Network (MSDN)

The Microsoft Developer Network at <http://msdn.microsoft.com> is home to all Microsoft software-development tools and technologies. This is where the hardcore computer geeks go to get information about Microsoft developer products.

HTML Home Page

All Web pages are formatted using HyperText Markup Language (HTML). The W3C (World Wide Web Consortium) is the “official” site for all things HTML (including that useful new mutant, XHTML). To go straight to documentation on HTML or XHTML, use the link www.w3.org/MarkUp/.

Cascading Style Sheets Home Page

Cascading Style Sheets (CSS) are a must if you intend to give your Web site a consistent look and feel. They're even more indispensable if you intend to use themes, because every theme includes some CSS. The home page for CSS is at www.w3.org/Style/CSS/.

XML Home Page

XML (eXtensible Markup Language) is a standardized means of transmitting raw data across the Internet via the World Wide Web. (By "raw" I mean unformatted, like the data in a database — useful even if it isn't pretty to look at.)

Whether your site needs (or can even use) XML depends on many factors. In Visual Web Developer, you may be able to get by with little or no XML at all — that depends on your site. If you do need XML info, the official site at www.w3.org/XML/ contains all the specs.

ASP.NET

If I was forced to pin down my choice for the main technology behind Visual Web Developer, ASP.NET 2.0 would be the one I'd choose. Although there are plenty of Web sites that address this topic, the most "official" would have to be the one whose URL is also the easiest in the world to remember:

<http://ASP.NET>.

ASP.NET Starter Kits

Clicking the Starter Kits tab at the ASP.NET Web site takes you to a page of sample Web applications that you can download for free. There are different kinds of Starter Kits for different types of Web sites. If you have a particular type of site in mind, but don't quite know how to organize things, a Starter Kit could be a great way to get started.

ASP.NET Forums

The ASP.NET Forums at <http://forums.asp.net> give you a handy place to go for questions and answers. There are a lot of discussions to choose from. You may have to scroll down to find Visual Web Developer and related discussions. You can discover a lot about what the technology can do just by reading through some of the questions and answers that have already been posted.

SQL Server Developer Center

The subtitle to this Web page, “Enabling a world of data-driven applications,” is what SQL Server is all about. In a Visual Web Developer site, SQL Server contains all the data that drives the Web site. As I write this, the URL for SQL Server 2005 is <http://msdn.microsoft.com/SQL/2005>. If that doesn’t work, try the more general URL <http://msdn.microsoft.com/SQL>.

dotnetjunkies

The Microsoft .NET Framework is a huge collection of pre-written software that most modern programmers use to create modern programs. In fact, all the Using statements you see at the top of a code-behind page in VWD are references to namespaces within the .NET Framework. Each namespace is like a folder containing usable code organized into classes.

The dotnetjunkies site (at www.dotnetjunkies.com) is a place for people who are into the whole .NET way of doing things. Though it spans the whole of .NET Framework development tools, there’s always info on ASP.NET 2.0 and Visual Web Developer.

Microsoft Technical Communities

This site encompasses all of Microsoft’s software-development tools — and discusses them in blogs, technical chats, newsgroups, webcasts, communities, user groups, and forums. There are all sorts of ways to interact with other people who use the same tools you’re using. The URL for this site is www.microsoft.com/communities.

Appendix

Publishing Your Site

In This Appendix

- ▶ Choosing a hosting provider
 - ▶ Preparing to upload
 - ▶ Uploading to the Web server
-

When all your site's Web pages are working properly, you're ready to upload your site to a *hosting provider* — a company that can host your Web site in a way that allows anybody with an Internet connection to view your site.

When you've chosen a hosting provider, exactly how you upload pages to the site will depend on the provider you chose. There is no simple one-rule-fits-all technique that I can give you. Only your provider can give you specific details on what's needed. But I can tell you, in general, what the process is all about.

Choosing a Hosting Provider

There are tons of companies out there who will be more than happy to host your Visual Web Developer Web site — for a fee, of course. (You may be able to get a few months free as a trial period, but that's normally temporary.) The ASP.Net Web site (www.ASP.net) provides links to many companies that offer that service.

When you're shopping for a hosting provider, you'll come across a ton of options to choose from. Your goal is to find a service that specifically supports Visual Web Developer and SQL Server.

You'll need a domain name, too. That *domain name*, preceded by `http://www`, becomes your site's URL. For example, if you get the domain name `DunceSchool.com`, then people will have to type `www.DunceSchool.com` into their browser's Address bar to visit your site.

You can set up your account and get a domain name at the same time. But there is no guarantee that the domain name you want is available. You might have to try several domain names before you find one that's available. You can do so by going to any site that offers help with selecting domain names (such as www.NetworkSolutions.com).

After you have selected a domain name and a hosting provider, you'll have the information needed to publish your site. For the sake of example, let's say you set up an account with the following choices:

- ✓ **Domain Name:** *DunceSchool.com*
- ✓ **UserName:** *YourUserName*
- ✓ **Password:** *YourPassword*



Don't forget your own Web site's domain name, username, and password. Print or jot down that information as soon as you get it. When you have a place to post your site, you also have two locations for the site:

- ✓ **Local:** The copy of the site that's on your computer, and has been since Day One.
- ✓ **Remote:** The copy of the site on the Web server that the rest of the world can browse to.

To make your site visible to the world at large, you copy it from your local PC to the remote Web server. If you edit, refurbish, or otherwise tweak your site, the place to do that is your local PC. Then you can upload the revised version.

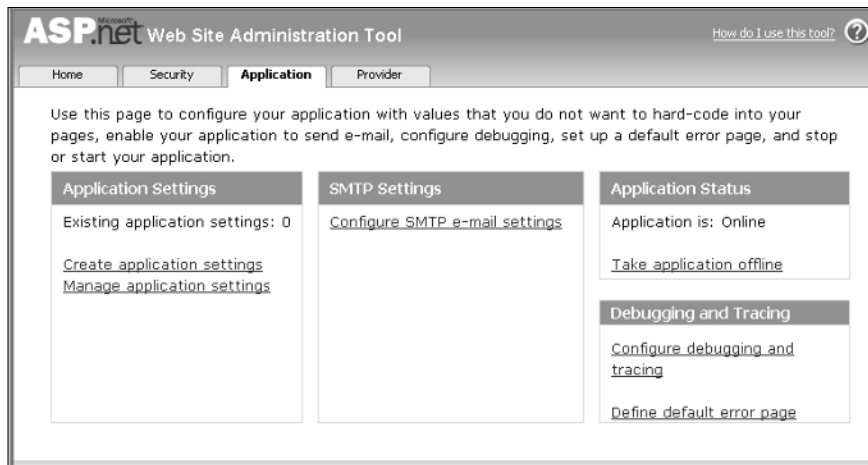
Preparing Your Site for Uploading

Once you have a place to upload your site, your ISP can fill you in on any specific tasks that have to be done prior to uploading. It's a good bet you'll want to turn off debugging and tracing in your site's `Web.config` file before uploading (more about why in a moment). To do that, follow these steps:

1. **Choose Website** ⇨ **ASP.NET Configuration** from the menu bar.
2. **In the Web Site Administration Tool, click Application Configuration.**

The options shown in Figure A-1 appear.

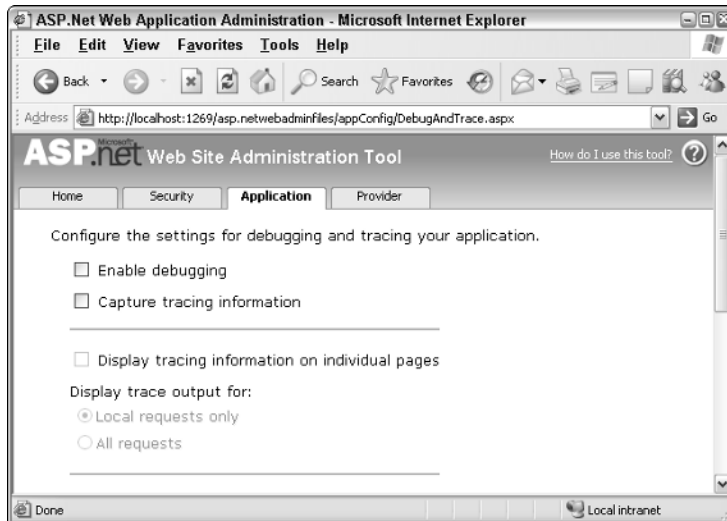
Figure A-1:
Application
Configuration in the
Web Site Adminstra-
tion Tool.



On a Web server, you generally don't want debugging and tracing enabled because those things eat up resources. The drain is trivial and unimportant on your local PC, but on a Web server — where the system is likely to be much busier — the effect of that resource consumption is magnified greatly.

So before you upload, click the [Configure debugging and tracing](#) link. Then clear the Enable Debugging and Capture Tracing Information check boxes, as shown in Figure A-2.

Figure A-2:
Turning off
debugging
and tracing.





Remember, I'm just showing you examples here, not specific instructions for your hosting provider. Only your hosting provider can give you specific instructions to configure your site for the services they provide.

If your site will send e-mail automatically, you also need to configure SMTP e-mail settings, using the link by that same name shown in Figure A-1). Again, only your hosting provider can tell you exactly how to configure your SMTP settings.

Copying the Site

After you have the above ducks all in a row, you're ready to copy your site to the hosting provider's server. You start by opening your Web site in Visual Web Developer, of course. Exactly how you do that will vary from one hosting provider to the next. But here are the general steps using FTP to upload pages:

1. Choose **Web Site** ⇨ **Copy Web Site** from the menu bar.
2. Click the **Connect** button near the top of the page that opens.
3. In the left column, click **FTP Site**.
4. Type the **FTP URL** — exactly as provided by your ISP — into the **Server** box.
5. Deselect the **Passive Mode** and **Anonymous Login** check boxes.
6. Type your **user name** and **password** into the appropriate boxes, as shown in Figure A-3, and click **Open**.

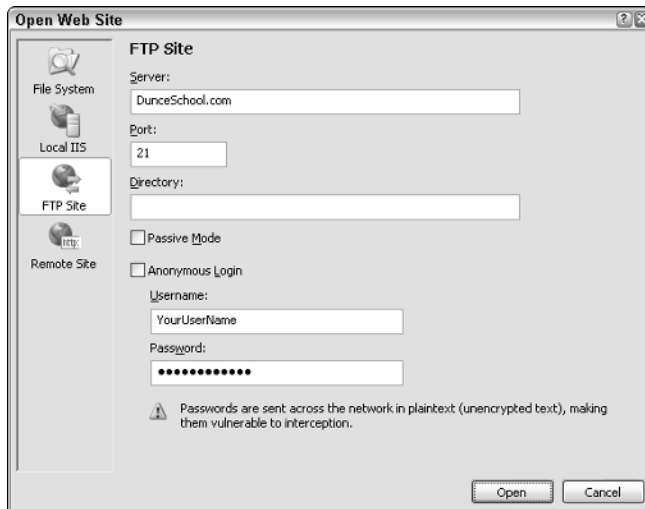


Figure A-3:
Turn off
debugging
and tracing.

When you're connected, the left pane shows your local site's files; the right pane shows the remote site's files. Initially, the remote site is empty. At this point, you simply want to copy all the files from your local site to the remote site. You can do so by clicking the button with the two-headed arrow between the two panes. Then you wait.

When all the files have been copied, the two columns will look the same, because the remote site contains all the same folders and files as the local site. To test the remote site, close Visual Web Developer. Then browse to the site's URL, just as if you were any user who wanted to visit the site.

If there's a problem with the remote site, chances are it's something you have to fix at that end — using a Control Panel or something similar on the remote site to do that final tweaking. But exactly what final tweaking is required is (you guessed it) information you can get only from your service provider.

Given that your local site — the one that lives on your PC — is the best place to tweak, you may want to re-enable debugging and tracing in that site only. Now, because that change will be stored in the site's `Web.config` file, you must remember *not* to upload that file to the remote site. In the Copy Files text box, the discrepancy between the local-site and remote-site `Web.config` files will show up as a pair of question marks, as shown in Figure A-4.

Figure A-4:

The local and remote `Web.config` files don't match — but that may be okay.

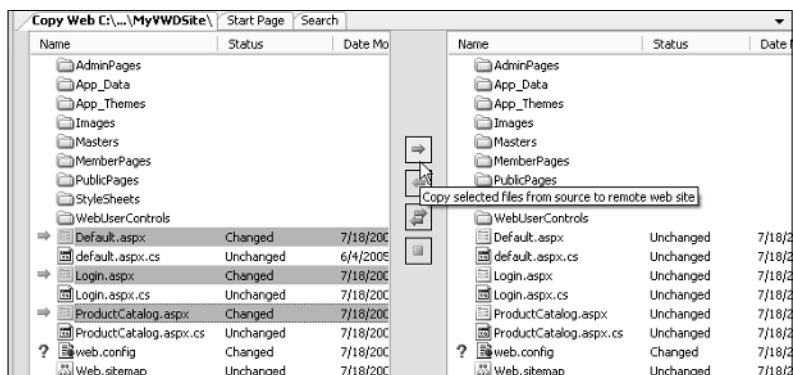
ProductCatalog.aspx	Unchanged	7/18/2005 8:45	ProductCatalog.aspx	Unchanged	7/18/2005 8:45
ProductCatalog.aspx.cs	Unchanged	7/18/2005 8:45	ProductCatalog.aspx.cs	Unchanged	7/18/2005 8:45
? web.config	Changed	7/18/2005 8:42	? web.config	Changed	7/18/2005 8:42
Web.sitemap	Unchanged	7/18/2005 8:45	Web.sitemap	Unchanged	7/18/2005 8:45

Don't presume that question marks mean "You must upload this file right now." The question marks simply mean that the two files are different. But in this case, you may very well *want* the two `Web.config` files to be different because the sites are on two different computers, requiring two different configuration files. Any *other* changes you make to your site, however, should be uploaded to the remote site as needed. Those changes show up as a blue arrow.



If you use the two-headed arrow box to copy *all* the pages from your local site to the remote computer in that case, you end up copying the local `Web.config` file too. To play it safe, select only the files you do want to copy, then click the right-pointing arrow button, as shown in Figure A-5, to copy just the modified pages without copying the modified `Web.config` file.

Figure A-5:
Copying just
the selected
files.



In a nutshell, that's how uploading pages to a Web server works in Visual Web Developer Express. In the non-Express version of the product, you also have the option to *publish* the site. The main difference between the two processes is that when you publish, you must first create a set of executable files, and then copy those files to the server.

What's on the CD-ROM?

This book includes two CD-ROMs. The first disc contains Visual Web Developer 2005 Express Edition software. The second disc contains supplemental material for Visual Basic 2005 Express and Visual Web Developer 2005 Express. The full contents of this second disc are discussed in the sections that follow.

Visual Basic 2005 Express

✓ **Videos:** Absolute Beginner's Guide to Visual Basic Express

This video series is designed specifically for individuals who are interested in learning the basics of how to create applications using Visual Basic 2005 Express Edition. This includes over *8 hours* of video-based instruction that walks from creating your first “Hello World” application to a fully functioning RSS Reader application. Learn how to write your first application today.

For more information on software development with Visual Basic Express Edition, you may be interested in these Supplemental Readings. (They require Adobe Acrobat Reader.)

Includes all 16 lessons with 17 videos and 12 downloads.

✓ **Starter Kits:** Starter Kits are enhanced project templates that can be shared with other members of the community. Here are some starter kits for Visual Basic.

- **Card Game Starter Kit:** This Starter Kit is a complete Black Jack card game. The starter kit contains an extensible framework for building card games and a Black Jack game application that is built on top of this framework. The project comes ready to compile and run, but it's easy to customize with only a little extra programming. The section Expanding the Card Game contains a list of some customizations you might make. You are also free to use the source code as the basis for your own card game projects, and share your work with others or upload it to the Internet.
- **Amazon-Enabled Movie Collection Starter Kit (link):** The Amazon-Enabled Movie Collection Starter Kit is a Windows Form application that uses Amazon.com's Web services to dynamically search for movie titles. This Starter Kit demonstrates technologies such as: calling XML Web services, databinding, application settings, local data storage using SQL Server 2005 Express Edition, and more.

- ✓ **Additional Resources:** Here are additional resources on the Web. Most of these links will be updated in the future, so you may want to occasionally check them for updated information and resources.
 - **Visual Basic Express Edition Home Page:** This page on the Microsoft Web site provides additional information and links for Visual Basic Express.
 - **Visual Basic Developer Center:** Visit the Microsoft Visual Basic Developer Center. Here you will find the most recent information on Visual Basic.
 - **Visual Basic Forums:** Read and place postings on the many ASP.NET forums.
 - **SQL Server Query Basics:** Learn to use the powerful T-SQL language and see how easy and flexible it is for retrieving information stored in SQL Server. This article introduces you to T-SQL and its robust query syntax that makes it easy for you to use SQL Server Express for your data-driven applications.

Visual Web Developer 2005 Express

- ✓ **Videos:** Learn Visual Web Developer Series

This video series is designed specifically for individuals who are interested in learning the basics of how to create dynamic Web applications using ASP.NET 2.0 and Visual Web Developer 2005 Express Edition in either Visual Basic or C#.

Includes all 14 lessons with 14 videos and 14 downloads for Visual Basic

Includes all 14 lessons with 14 videos and 14 downloads for C#
- ✓ **Videos:** Videos by Jeff Prosis of Wintellect (Links to MS Web Site)

This series of video presentations by Jeff Prosis of Wintellect introduces you to many of the new features of ASP.NET 2.0, such as:

 - **Web Forms:** Web forms are the atoms from which ASP.NET Web pages are built. This module introduces the Web forms programming model and acquaints developers with some of the Web controls featured in ASP.NET.
 - **State Management:** ASP.NET 2.0 provides a variety of mechanisms for building stateful Web applications. This module introduces them all: view state, application cache, session state, profiles, and cookies.

- **Security:** Learn the essentials of ASP.NET security with an emphasis on forms authentication, the membership and role management services, and login controls.
 - **Master Pages and Site Navigation:** Get an introduction to master pages, which are templates that provide content to other pages, and to the data-driven site navigation tools featured in ASP.NET 2.0.
 - **Data Access:** New controls in ASP.NET 2.0 enable developers to build sophisticated, data-driven Web pages with little or no code. This module introduces developers to data-driven content, ASP.NET-style.
 - **Application Infrastructure:** Pages alone do not a Web site make. This module introduces other essential elements of ASP.NET applications, including Web.config, Global.asax, components, and resources.
- ✔ **Starter Kits:** The ASP.NET 2.0 Starter Kits for Visual Web Developer are fully functional sample applications to help you learn ASP.NET 2.0. Each sample is complete and well-documented so that you can use the code to kick start your Web projects today.
- **Club Site Starter Kit (link):** This starter kit provides a starting point for creating a Web site for your club or organization. The kit includes news posting, calendaring, member directory, and photo album systems.
 - **Time Tracker Starter Kit (link):** This is a business Web application for keeping track of hours spent on a project, with the ability to handle multiple resources as well as multiple projects.
- ✔ **Additional Resources:** Here are additional resources on the Web. Most of these links will be updated in the future, so you may want to occasionally check them for updated information and resources.
- **Visual Web Developer 2005 Express Edition Guided Tour:** Learn about the key new features available in Visual Web Developer 2005 Express that will make Web development easier and more productive than ever before.
 - **ASP.NET 2.0 QuickStart Tutorial:** The ASP.NET QuickStart is a series of ASP.NET samples and supporting commentary designed to quickly acquaint you with the syntax, architecture, and power of the ASP.NET Web programming framework. The QuickStart samples are designed to be short, easy-to-understand illustrations of ASP.NET features. By the time you finish reading this tutorial, you will be familiar with the broad range of the new features in ASP.NET 2.0, as well as the features that were supported in earlier versions.

- **ASP.NET:** Visit Microsoft's Web site for the latest information on ASP.NET.
- **ASP.NET Developer Center:** The Official Microsoft Developer Center for ASP.NET
- **Visual Web Developer 2005 Express Edition Home Page:** This page on the Microsoft Web site provides additional information and links for Visual Web Developer Express.
- **ASP.NET Forums:** Read and place postings on the many ASP.NET forums.
- **SQL Server Query Basics:** Learn to use the powerful T-SQL language and see how easy and flexible it is for retrieving information stored in SQL Server. This article introduces you to T-SQL and its robust query syntax that makes it easy for you to use SQL Server Express for your data-driven applications.

Index

• Symbols •

- : (colon) character, declarations, 99
- { and } (curly braces) characters, CSS style elements, 99
- . (period) character, class indicator, 103
- ;(semicolon) character, declarations, 99

• A •

- absolute positioning
 - enabling, 193
 - object alignments, 195–196
 - Z-indexes, 194–195
- access rules, membership sites, 45–48
- Active Server Pages (ASP),
 - described, 322
- Add New Item dialog box
 - adding style sheets to a theme, 202–203
 - CSS style sheet creation, 100–101
 - new blank page creation, 76
 - site map addition, 158
 - Web form addition, 176
 - Web Form creation, 205
 - Web User Control addition, 166–167
- Add New Style Rule dialog box, class rule creation, 103
- Add ORDER BY Clause dialog box, data-bound control sorts, 267–268
- Add Style Rule dialog box
 - CSS class selector creation, 103
 - CSS style rule creation, 102
- Add Table dialog box, view creation, 249
- Add WHERE Clause dialog box, data-bound control query filters, 270
- alignments
 - absolutely-positioned objects, 195–196
 - HTML table cell text, 82
- anonymous users
 - membership site visits, 39–40
 - security trimming, 161–162
- App_Data folder, default database storage, 22
- App_Themes folder
 - subfolders, 200–201
 - theme storage, 199–200
- applications, described, 321
- ASP (Active Server Pages),
 - described, 322
- ASP.NET. *See also* controls
 - ChangePassword control, 145–146
 - control/template conversion, 128–129
 - forums, 329
 - Login controls, 130–131
 - login links, 136–141
 - Login.aspx page creation, 135–136
 - LoginName control, 138
 - LoginStatus control, 137–138
 - LoginView control, 138–141
 - membership testing, 146–148
 - Microsoft .NET Framework integration, 123, 135
 - password constraints, 149–151
 - PasswordRecovery control, 141–144
 - server control addition, 125–130
 - specifications, 328
 - Starter Kits, 328
 - template editing, 129–130
 - user account creation, 131–135
 - viewing server controls in Source view, 148–149
 - workflow process, 123–124

- ASP.NET 2.0 Hosters, publishing to the Web, 19
- ASP.NET 2.0, described, 322
- aspnet_Profile table, profile property storage, 187
- ASPNETDB.MDF database, Solution Explorer display, 51–52
- attributes, Source view entry conventions, 93–95
- authentication
 - Forms authentication, 42–43
 - membership sites, 42–43
 - user profiles, 171–175
 - validation controls, 188–192
 - Windows authentication, 42–43
- Auto Format, server controls, 127

• B •

- background colors
 - cells, 82
 - Master Pages, 57–58
- backgrounds, styling, 107–108
- binary data type, SQL Server, 228
- binding
 - DetailsView control, 287–290
 - DropDownList control, 280–287
 - navigation controls, 163–164
- bit data type, SQL Server, 232
- blank pages, creating, 75–77
- bookmarks
 - creating, 86
 - links, 86–87
- Boolean data type, SQL Server, 228, 232
- borders
 - cells, 59–60
 - HTML table cells, 82–83
 - pictures, 89–91
 - styling, 113–114
- boxes, styling, 113–114

- browsers
 - compatibility issues, 17–19
 - viewing Web pages, 65–66
 - Web page viewing, 35–36
- Button control, adding to a Web form, 178–179
- buttons, adding to a Web form, 178–179

• C •

- C# language, supported programming language, 21
- captions, picture, 307–308
- Cascading Style Sheets (CSS). *See also* styles; style sheets
 - class selectors, 102–104, 117–119
 - declaration component, 99–100
 - described, 97–100
 - DIV styles, 120–121
 - element class selectors, 119–120
 - element selectors, 116–117
 - element styles, 101–102
 - HTML interaction, 97–100
 - saving style sheets, 115
 - selector component, 99–100
 - specifications, 328
 - Style Builder editing, 57, 104–115
 - style rules, 101–104
 - style sheet creation, 100–101
 - style sheet links, 115–116
 - theme element, 201–204
 - toolbar viewing, 100–101
 - W3C specifications, 122
- cell borders, Master Pages, 59–60
- cells
 - background, 82
 - borders, 82–83
 - control addition, 84
 - deleting, 79
 - height/width settings, 58–59
 - HTML table insertion, 79
 - HTML table merges, 80–81

- HTML table selections, 80
 - resizing, 79
 - selections, 80
 - styling, 81–83
 - table data entry, 78
 - text alignments, 82
 - text formatting, 82
- ChangePassword control, ASP.NET, 145–146
- ChangePassword, Login control, 131
- char data type, SQL Server, 230
- character data, text data type, 228–230
- class indicator, period (.), 103
- class selectors
 - CSS (Cascading Style Sheets), 102–104
 - Web page application, 117–118
- code-behind files
 - tying code to an event, 180–183
 - viewing, 33–34
- codes
 - conventions used in book, 3
 - described, 323–324
 - elements, 323–324
- colon (:) character, declarations, 99
- colors
 - background, 57–58
 - background styling, 107–108
 - cell background, 82
- columns
 - adding to HTML tables, 79
 - data-bound control retrieval, 265–267
 - DataList control picture display, 305–306
 - DataList control text display, 301–302
 - date/time number formatting, 274–275
 - deleting, 79
 - grid arrangements, 279–280
 - hiding/displaying in grids 279–280
 - selections, 79–80
 - SQL Server table element, 222–223
- Common Tasks menu, server controls, 127–130
- CompareValidator control, text box value comparison, 191–192
- compatibility, Web browser settings, 17–19
- components, book sections, 4
- composite formatting, date/time number formatting, 274–275
- connections, data-bound controls, 262–264
- content pages, creating, 65
- ContentPlaceHolder pane, Master Pages, 55–56, 61–63
- contents
 - adding to Master Pages, 67–68
 - Help system element, 15
- controls. *See also* ASP.NET; Login controls; server controls
 - absolute positioning, 193–198
 - adding to Master Pages, 68
 - data binding methods, 262–274
 - described, 323
 - grid display, 276–280
 - HTML table cell addition, 84
 - login links, 136–141
 - programmatic interaction method, 324–325
 - programmatic name assignment, 177
 - server, 125–130
 - site-navigation, 154–158
 - skin files, 201, 204–209
 - static versus dynamic data, 155
 - template conversion, 128–129
 - template editing, 129–130
 - theme selection, 212–213
- conventions, used in book, 3
- CreateUserWizard, Login control, 131–135
- CSS (Cascading Style Sheets). *See also* styles; style sheets
 - class selectors, 102–104, 117–119
 - curly braces ({ and }) characters, 99
 - declaration component, 99–100

CSS (*continued*)

- described, 97–100
- DIV styles, 120–121
- element class selectors, 119–120
- element selectors, 116–117
- element styles, 101–102
- HTML interaction, 97–100
- saving style sheets, 115
- selector component, 99–100
- specifications, 328
- Style Builder editing, 57, 104–115
- style rules, 101–104
- style sheet creation, 100–101
- style sheet links, 115–116
- theme element, 201–204
- toolbar viewing, 100–101
- W3C specifications, 122
- CSS Editor, style sheet display, 100–101
- curly braces { and } characters, CSS style elements, 99
- CustomValidator control, form validation, 192

• D •

- data, binding to controls, 262–274
- Data Configuration Wizard
 - binding data to a control, 262–273
 - column retrieval, 265–267
 - connection selections, 262–264
 - data source selections, 262–264
 - filtering, 269–273
 - query filters, 269–273
 - Select Statement configuration, 264
 - sort order settings, 267–268
 - table selections, 265
 - unique value display, 268–269
 - view selections, 265
- data sources, data-bound controls, 262–264

data types

- binary, 228
- bit, 232
- Boolean, 228, 232
- char, 230
- date/time, 228, 232
- datetime, 232
- decimal, 231
- float, 231
- int, 231
- money, 231
- nchar, 230
- number, 228, 230–232
- numeric, 231
- nvarchar (MAX), 230
- nvarchar, 230
- other (specialized), 228, 232–233
- real, 231
- smalldatetime, 232
- smallmoney, 231
- SQL Server, 227–233
- table definitions, 227–228
- text, 228–230
- theme information, 201
- tinyint, 231
- user profiles, 172–175
- varchar (MAX), 230
- varchar, 230
- Database Explorer
 - table creation, 236–237
 - user interface element, 11
 - view creation, 249
 - viewing existing tables, 222–223
 - viewing table definitions, 227–228
- database management system (DBMS), described, 325–326
- databases
 - ASPNETDB.MDF, 51–52
 - data-bound control data sources, 262–264
 - described, 325–326

- drop-down list data binding, 280–287
 - hyperlinks table, 257–259
 - many-to-many relationships, 223–227
 - money fields, 240
 - one-to-many relationships, 223–227
 - picture table, 254–257
 - primary key field, 237–238, 241–244
 - SQL Server storage advantages, 221
 - SQL Server table elements, 222–223
 - table data entry, 244–246
 - table links, 247–254
 - text fields, 238–240
 - Transactions table, 241–244
 - user access prevention techniques, 275–276
 - data-driven, described, 1, 322
 - DataList control, normal text data display, 296–312
 - date/time columns, date/number formatting, 274–275
 - date/time data type, SQL Server, 228, 232
 - dates
 - DataList control formatting, 300–301
 - formatting, 274–275
 - GridView control formatting, 279
 - uniqueidentifier, 232–233
 - datetime data type, SQL Server, 232
 - DBMS (database management system), described, 325–326
 - debugging, HTML in Source view, 95–96
 - decimal data type, SQL Server, 231
 - declarations
 - colon (:) character, 99
 - CSS component, 99–100
 - semicolon (;) character, 99
 - Default.aspx, default Web page, 22–23
 - Design Surface
 - How Do I page display, 15
 - page editing techniques, 26–31
 - saving changes before closing, 32–33
 - title creation, 34–35
 - user interface element, 10–11
 - viewing code-behind files, 33–34
 - Design view
 - absolute positioning, 193–194
 - Common Tasks menu, 127–130
 - data-bound controls, 273–274
 - LoginStatus control display, 140
 - moving objects, 56
 - server control editing, 126–127
 - switching to Source view, 23, 30–31
 - Web page display, 23
 - DetailsView control
 - data display/editing, 287–290
 - Master-Details forms, 294–295
 - developers, described, 321
 - directory. *See* folders
 - DIV styles, Web page application, 120–121
 - domain name, Web publishing requirement, 331–332
 - dotnetjunkies, online resource, 329
 - drop-down lists, data binding, 280–287
 - DropDownList control
 - drop-down list binding, 280–287
 - Master-Details forms, 292–293
 - dynamic data, navigation control support, 155
 - Dynamic Help pane, Help system element, 15–16
 - dynamic, described, 1
- **E** ●
- element class selectors, Web page application, 119–120
 - element selectors, Web page application, 116–117
 - element styles, CSS (Cascading Style Sheets), 101–102
 - e-mail addresses, PasswordRecovery control element, 142–143

Error List pane, debugging HTML, 95–96
 eXtensible Markup Language (XML),
 specifications, 328
 eyebrow menus, adding to a Web page,
 164–165

• F •

false/true values, Boolean data type,
 228, 232
 favorites, Help system element, 15
 fields
 foreign keys, 241
 money, 240
 primary key, 237–238, 241–244
 RequiredFieldValidator control,
 189–190
 SQL Server table element, 222–223
 table data entry, 244–246
 text, 238–240
 files
 ASPNETDB.MDF, 51–52
 code-behind, 33–34
 copying to a folder, 25–26
 copying/pasting to a folder, 26
 dragging/dropping to a folder, 25–26
 Login.aspx, 135–136
 RecoverPassword.aspx, 144
 skin, 201, 204–209
 Web.config, 51–52, 150–151, 162,
 173–175
 Web.sitemap, 158–164
 filters
 data-bound control query, 269–273
 DropDownList control, 282–284
 float data type, SQL Server, 231
 floating-point numbers, scalar value
 type, 230–231
 folders
 App_Data, 22
 App_Themes, 199–200
 copying files to, 25–26

 creating, 24–25
 database access prevention
 techniques, 275–276
 deleting, 26
 dragging/dropping files to, 25–26
 hiding/showing contents, 26
 Master Pages, 54
 members-only content, 39–40
 renaming, 24, 26
 Theme, 199–201
 Font Picker, font selections, 106
 fonts
 sizing, 106–107
 styling, 105–107
 footers, Master Page layout, 53
 foreign keys, SQL Server tables, 241
 formatting, text, 27–28
 forms. *See also* Web forms
 absolutely-positioned objects, 193–198
 custom validation controls, 192
 failed validation summary, 192
 Master-Details page, 291–296
 regular expression validation control,
 190–191
 required field validation control,
 189–190
 text box validation control, 189–190
 text box value comparison, 191–192
 user profile information entry, 176–188
 validation controls, 188–192
 value entry validation control, 190
 Forms authentication, membership sites,
 42–43
 FormView control, record data display,
 312–318
 forums, ASP.NET, 329

• G •

Globally Unique Identifier (GUID), SQL
 Server, 232–234
 Google, search engine workflow process,
 123–124

grids, GridView control, 276–280
GridView control
 grid display, 276–280
 Master-Details forms, 293–294
 subtotal display, 314–318

• H •

headers, Master Page layout, 53
Help on Help pane, Help system
 element, 15
Help system, user interface element,
 14–16
Help toolbar, user interface element,
 14–15
home page, described, 327
horizontal rules, Web page addition, 92
hosting providers
 uploading sites to, 334–336
 Web publishing, 19, 331–332
 Web publishing preparations, 332–334
How Do I page, Design Surface display, 15
HTML pages
 CSS interaction, 97–100
 debugging in Source view, 95–96
 home page, 327
 uses, 75
HTML tables
 adding to a page, 77–78
 cell data entry, 78
 cell insertion/deletion, 79
 cell merges, 80–81
 cell selections, 80
 cell styling, 81–83
 column addition/deletion, 79
 control addition, 84
 deleting, 79
 resizing cells, 79
 row addition/deletion, 79
 row/column selections, 79–80
HyperLink control, picture captions,
 307–308

Hyperlink dialog box, converting text
 selections to a hyperlink, 85
hyperlinks. *See also* links
 adding to a Web page, 84–87
 DataList control display, 309–312
 quick links, 85–86
 SQL Server table, 257–259

• I •

icons, used in book, 5
IDE (Integrated Development
 Environment), described, 322–323
Image control
 picture captions, 307–308
 sizing pictures, 305
index, Help system element, 15
inline elements, styling, 108
Insert Table dialog box, adding HTML
 tables to a page, 77–78
int data type, SQL Server, 231
integers, scalar value type, 230–231
Integrated Development Environment
 (IDE), described, 322–323
IntelliSense menu
 ignoring, 31
 selections, 31–32
Internet, Forms authentication, 42–43

• J •

J# language, supported programming
 language, 21

• L •

Label control, creating, 194
layouts
 Master Pages, 53, 55–56
 styling, 112–113
 template selections, 55–56

lines (statements), code element,
323–324

lines, Web page addition, 92

links. *See also* hyperlinks

ASP.NET login, 136–141

bookmarks, 86–87

hyperlinks table, 257–259

picture captions, 307–308

picture table, 254–257

RecoverPassword.aspx page, 144

SQL Server tables, 247–254

style sheets, 115–116

list boxes, theme selections, 212–213

ListBox control, theme selections,
212–213

ListItem Collection Editor, theme list box
selections, 213

local networks, Windows authentication,
42–43

Login, Login control, 130

Login controls. *See also* controls

adding to a Web page, 130–131

types, 130–131

login links, ASP.NET, 136–141

Login.aspx page, creating, 135–136

LoginName control, ASP.NET, 138

LoginName, Login control, 131

LoginStatus control

ASP.NET, 137–138

Design view display, 140

LoginView control, ASP.NET, 130,
138–141

• M •

many-to-many relationships, SQL Server,
223–227

margins, styling, 113

Master Pages. *See also* Web pages

adding to existing Web pages, 69–71

background color, 57–58

cell borders, 59–60

cell height/width settings, 58–59

content addition, 67–68

content pages, 65

ContentPlaceHolder pane styling,
55–56, 61–63

creating, 54–55

editing techniques, 66–69

folder creation, 54

layouts, 53, 55–56

left pane styling, 60–61

LoginStatus control, 137–138

LoginView control addition, 139–140

moving objects, 56

panes, 55–56

picture display troubleshooting, 71

Style Builder, 57–63

style sheet links, 115–116

template selections, 55–56

text alignments, 58

theme application, 217–218

Toolbox control addition, 68

Web forms, 63–66

Master-Details forms

DetailsView control, 294–295

DropDownList control, 292–293

GridView control, 293–294

transaction editing uses, 291–292

membership sites

access rules, 45–48

active versus inactive user accounts, 49

ASPNETDB.MDF database, 51–52

authentications, 42–43

GUID (Globally Unique Identifier),
232–234

members-only content folders, 39–40

preferred theme storage, 213–214

roles (people categorizes), 43–45

security trimming, 161–162

SQL Server connection testing, 41–42

testing in ASP.NET, 146–148

theme page display/selections, 211–212

theme selections, 210–216

user accounts, 39–40, 48–51
 viewing existing tables, 222–223
 WAT (Web Site Administration Tool),
 40–43
 Web.config file, 51–52
 members-only content, folder creation,
 39–40
 Menu control, site-navigation, 154–158
 Menu Item Editor dialog box, navigation
 control addition, 156–158
 merges, HTML table cells, 80–81
 Microsoft .NET Framework, ASP.NET
 integration, 123, 125
 Microsoft Access, DBMS (database
 management system), 325–326
 Microsoft Developer Network (MSDN),
 online help resource, 327
 Microsoft, technical communities, 329
 money data type, SQL Server, 231
 money fields, SQL Server tables, 240

• **N** •

NavigateUrl property, navigation
 controls, 156–158
 navigation controls
 binding to Web.sitemap file, 163–164
 dynamic data support, 155
 eyebrow menus, 164–165
 security trimming, 161–162
 site maps, 158–164
 static data support, 155
 ToolTips, 156–158
 nchar data type, SQL Server, 230
 New Web Site dialog box, new Web site
 creation, 22–23
 newspaper-style columns, DataList
 control text display, 301–302
 non-Unicode text, text data type, 229–230
 <not set> setting, Style Builder, 104
 null value, Boolean data type, 228, 232

number data type, SQL Server, 228,
 230–232
 numbers
 DataList control formatting, 300–301
 formatting, 274–275
 subtotal display, 314–318
 numeric data type, SQL Server, 231
 nvarchar (MAX) data type, SQL
 Server, 230
 nvarchar data type, SQL Server, 230

• **O** •

objects
 alignments, 195–196
 moving, 56
 property editing, 29–30
 sizing, 196–197
 spacing, 197–198
 stacking, 194–195
 one-to-many relationships, SQL Server,
 223–227
 online resources, Web sites, 16–17
 other (specialized) data type, SQL
 Server, 228, 232–233

• **P** •

padding, styling, 113
 pages. *See* Web pages
 panes
 ContentPlaceHolder, 55–56, 61–63
 docking/undocking, 12
 hiding/displaying, 12
 Master Page layout element, 55–56
 Master Page styling, 60–63
 moving objects, 56
 moving/sizing, 11–12
 resetting to default display, 12
 View menu options, 13–14

- Parameter Values Editor dialog box,
 - data-bound control query testing, 272
 - PasswordRecovery control, ASP.NET, 141–144
 - PasswordRecovery, Login control, 131
 - passwords
 - authentication method, 42–43
 - ChangePassword control, 145–146
 - constraint editing, 149–151
 - PasswordRecovery control, 141–144
 - user accounts, 48–49
 - people categories (roles)
 - access rules, 45–48
 - membership sites, 43–45
 - period (.) character, class indicator, 103
 - permissions, access rules, 45–48
 - phone numbers, user profile element, 172
 - pictures
 - adding to a Web page, 28–29
 - binary data type, 228
 - borders, 89–91
 - captions, 307–308
 - centering troubleshooting, 91
 - DataList control display, 302–308
 - Master Page display troubleshooting, 71
 - padding, 91
 - position styling, 110–111
 - positioning, 88–89
 - sizing, 88, 305
 - SQL Server table, 254–257
 - styling, 88–91
 - text wrapping, 89–90
 - theme element, 201–202
 - watermark uses, 107–108
 - Web page addition, 87–91
 - Pointer, Login control, 130
 - postbacks, when to execute, 185
 - precision, decimal data type limits, 231
 - primary keys
 - SQL Server tables, 237–238, 241–244
 - Transactions table, 243–244
 - profile properties
 - default values, 172, 174
 - information storage, 187
 - information types, 172, 173–175
 - retrieving/editing, 184–187
 - user profiles, 171
 - viewing/editing, 284–287
 - Visual Basic, 187–188
 - programmatically names, control assignment, 177
 - programmatically, control interaction method, 324–325
 - programming languages
 - preference selections, 22
 - supported types, 21
 - properties
 - object editing, 29–30
 - user interface element, 11
 - validation controls, 188–192
 - viewing/editing, 284–287
 - Properties pane, object editing, 29–30
 - Properties sheet
 - docking/undocking, 29
 - object property editing, 29–30
 - publishing to the Web
 - copying sites to hosting providers, 334–336
 - domain name requirement, 331–332
 - hosting providers, 331–332
 - preparing for uploads, 332–334
 - services, 19
- *Q* •
- queries
 - data-bound control filters, 269–273
 - SQL Server views, 248–254
 - Query Builder, SQL Server views, 248–254
 - quick links, creating, 85–86

• R •

RangeValidator control, value entry, 190

RDBMS (relational database-management system), SQL Server, 223

readers, author's assumptions, 2–3, 9–10

real data type, SQL Server, 231

records

- DropDownList control filtering, 282–284
- FormView control display, 312–318
- SQL Server table element, 222–223

RecoverPassword.aspx page, testing, 144

regular expressions, validation control, 190–191

RegularExpressionValidator control, validation criteria, 190–191

relational database-management system (RDBMS), SQL Server, 223

relationships

- many-to-many, 223–227
- one-to-many, 223–227

RequiredFieldValidator control, text boxes, 189–190

resources, online, 16–17

Results pane, query results display, 250–251

rolegroups, creating/editing, 141

roles (people categories)

- access rules, 45–48
- membership sites, 43–45

rows

- adding to HTML tables, 79
- data-bound control query filters, 269–273
- deleting, 79
- selections, 79–80
- SQL Server table element, 222–223

• S •

scalar values

- floating-point numbers, 230–231
- integers, 230–231
- math function support, 172
- number data type, 228

scale, decimal data type limits, 231

schemes, server controls, 127–128

search engines, workflow process, 123–124

searches

- Help system element, 15
- user accounts, 50

security

- access rules, 45–48
- authentications, 42–43
- user access prevention techniques, 275–276

security trimming, anonymous users, 161–162

Select a Master Page dialog box, 64

Select Statements, data-bound control configuration, 265

selections

- converting to a hyperlink, 85
- HTML tables rows/columns, 79–80
- text, 27–28
- themes, 210–216

selectors, CSS component, 99–100

semicolon (;) character, declarations, 99

server controls. *See also* controls

- Auto Format, 127
- Common Tasks menu, 127–130
- editing in Design view, 126–127
- schemes, 127–128
- viewing in Source view, 148–149
- Web page addition, 125–130

sidebars, Master Page layout, 53

site maps

- adding to a Web page, 158–161
- dynamic data support, 155

- SiteMapPath control, adding
 - breadcrumb menu to a Web page, 164–165
- sizes, absolutely-positioned objects, 196–197
- skins
 - default versus named, 207–209
 - described, 201
 - file creation, 204–207
 - theme element, 201
- smalldatetime data type, SQL Server, 232
- smallmoney data type, SQL Server, 231
- Solution Explorer
 - adding a picture to a Web page, 28–29, 87
 - ASPNETDB.MDF database display, 51–52
 - code-behind files, 33–34
 - copying files to a folder, 25–26
 - default theme creation, 199–200
 - folder creation, 24–25
 - log in before viewing restricted page, 182
 - Master Pages folder creation, 54
 - members-only content folder creation, 40
 - new blank page creation, 76–77
 - opening Web pages, 24
 - quick link creation, 85–86
 - renaming folders, 24
 - user interface element, 11
 - viewing Web pages in a browser, 65–66
 - Web form creation, 64
 - Web User Controls, 165–167
 - Web.config file display, 51–52
- sorts, data-bound controls, 267–268
- Source view
 - absolutely-positioned item display, 195
 - bookmark display, 86
 - editing techniques, 92–96
 - HTML debugging, 95–96
 - selection techniques, 93
 - server control viewing, 148–149
 - switching to Design view, 23, 30–31
 - tag/attribute entry conventions, 93–95
 - Web page display, 23
 - Web page editing techniques, 31–32
- spacing, absolutely-positioned objects, 197–198
- specialized (other) data type, SQL Server, 228, 232–233
- SQL Server
 - binary data type, 228
 - bit data type, 232
 - Boolean data type, 228, 232
 - char data type, 230
 - columns, 222–223
 - data storage advantages, 221
 - data type summary listing, 234–236
 - date/time data type, 228, 232
 - datetime data type, 232
 - DBMS (database management system), 325–326
 - decimal data type, 231
 - development center, 329
 - float data type, 231
 - foreign keys, 241
 - GUID (Globally Unique Identifier), 232–234
 - hyperlinks table, 257–259
 - int data type, 231
 - many-to-many relationships, 223–227
 - membership site connection testing, 41–42
 - money data type, 231
 - money fields, 240
 - nchar data type, 230
 - non-Unicode text, 229–230
 - number data type, 228, 230–232
 - numeric data type, 231
 - nvarchar (MAX) data type, 230
 - nvarchar data type, 230

- one-to-many relationships, 223–227
- other (specialized) data type, 228, 232–233
- picture tables, 254–257
- primary keys, 237–238, 241–244
- Query Builder, 248–254
- query views, 248–254
- RDBMS (relational database-management system), 223
- real data type, 231
- rows, 222–223
- saving tables, 240–241
- smalldatetime data type, 232
- smallmoney data type, 231
- statements, 247–248
- table creation, 236–237
- table data entry, 244–246
- table data types, 227–233
- table definitions, 227–228
- table links, 247–254
- tables, 222–223
- text data type, 228–230
- text fields, 238–240
- tinyint data type, 231
- Transactions table, 241–244
- Unicode text, 229–230
- uniqueidentifier data type, 232–233
- varchar (MAX) data type, 230
- varchar data type, 230
- Start Page, disabling/enabling, 14
- Starter Kits, ASP.NET, 328
- statements (lines)
 - code element, 323–324
 - SQL Server, 247–248
- static data, navigation control support, 155
- String data type, user profiles, 172–175
- strings
 - connections, 263
 - text data types, 229–230
- Style Builder
 - accessing, 30
 - background color, 57–58
 - background styling, 107–108
 - box/border styling, 113–114
 - cell borders, 59–60
 - cell height/width settings, 58–59
 - ContentPlaceHolder pane styling, 61–63
 - CSS (Cascading Style Sheets) specifications, 57
 - DIV styles, 121
 - font styling, 105–107
 - HTML table cell styling, 81–83
 - layout styling, 112–113
 - left pane styling, 60–61
 - <not set> setting, 104
 - saving changes, 114–115
 - stacking objects, 194–195
 - style rules, 104–115
 - styling position, 110–111
 - text alignments, 58, 108–110
 - white space styling, 108–110
- style rules
 - CSS (Cascading Style Sheets), 101–104
 - Style Builder editing, 104–115
- style sheets. *See also* Cascading Style Sheets (CSS)
 - adding to a theme, 202–204
 - copying between Theme folders, 203
 - creating, 100–101
 - links, 115–116
 - saving changes, 115
- styles. *See also* Cascading Style Sheets (CSS)
 - GridView control, 280
 - skin file creation, 205–206
- subtotals, Web page display, 314–318
- symbols, date/time number formatting, 275

• T •

table definitions, SQL Server tables, 227–228

tables

binary data type, 228

Boolean data type, 228, 232

closing, 223

column definitions, 236–237

creating, 236–237

data entry, 244–246

data information display, 222–223

data types, 227–223

data-bound control selections 265

date/time data type, 228, 232

DetailsView control binding, 287–290

foreign keys, 241

grid display control, 276–280

HTML, 77–84

hyperlinks, 257–259

links, 247–254

many-to-many relationships, 224–227

money fields, 240

number data type, 228, 230–232

one-to-many relationships, 224–227

other (specialized) data type, 228, 232–233

picture, 254–257

primary keys, 237–238, 241–244

profile property storage, 187

saving, 240–241

SQL Server organization, 222–223

styling, 110

table definitions, 227–228

text data type, 228–230

text fields, 238–240

Transactions, 241–244

viewing existing, 222–223

tabs, Help system navigation, 16

tags, Source view entry conventions, 93–95

templates

control conversion, 128–129

DataList control, 298–299

editing, 129–130

Master Page layouts, 55–56

text

adding to a Web page, 27

DataList control display, 296–300

formatting, 27–28

selections, 27–28

undoing changes, 28

Text property, navigation controls, 156–158

text alignments

Master Pages, 58

styling, 108–110

text boxes

CompareValidator control, 191–192

RequiredFieldValidator control, 189–190

text data type, SQL Server, 228–230

text fields, SQL Server tables, 238–240

text formatting, HTML table cells, 82

text wrapping, pictures, 89–90

Textbox controls

user profile information entry, 177–178

viewing user properties, 285–287

Theme folders

creating subfolders, 200–201

deleting, 200

renaming, 200

theme storage, 199–200

themes

application methods, 214–216, 218–219

data information types, 201

default creation, 199–200

described, 199

folders/subfolders, 199–201

ListBox control, 212–213

- Master Page application, 217–218
- member page display/selections, 211–212
- member selections, 210–216
- pictures, 201–202
- preferred storage, 213–214
- site-wide default theme, 219–220
- skins, 201, 204–209
- style sheets, 201, 202–204
- testing, 216
- Web page uses, 209–210
- time/date columns, number formatting, 274–275
- time/date data type, SQL Server, 228, 232
- times, GridView control formatting, 279
- tinyint data type, SQL Server, 231
- titles, creating, 34–35
- toolbars, viewing, 100–101
- Toolbox
 - adding controls to a HTML table cell, 84
 - adding controls to Master Pages, 68
 - adding server controls to a Web page, 125
 - navigation controls, 155–158
 - user interface element, 10–11
 - validation controls, 188–192
- ToolTips, navigation controls, 156–158
- Transactions table, SQL Server, 241–244
- transactions
 - Master-Details forms, 291–292
 - primary key field, 243–244
- TreeView control, site-navigation, 154–158
- troubleshooting
 - Master Page picture display, 71
 - picture centering, 91
- true/false values, Boolean data type, 228, 232

• u •

- Unicode text, text data type, 229–230
- uniqueidentifier data type, SQL Server, 232–233
- uploads
 - copying sites to hosting providers 334–336
 - Web publishing preparations, 332–334
- user accounts
 - active versus inactive users, 49
 - ASP.NET creation, 131–135
 - authentications, 42–43
 - CreateUserWizard control, 132–135
 - creating, 48–49
 - database access prevention
 - techniques, 275–276
 - editing/deleting, 50–51
 - GUID (Globally Unique Identifier), 232–234
 - many-to-many relationships, 224–227
 - membership sites, 39–40
 - membership testing, 146–148
 - PasswordRecovery control, 141–144
 - passwords, 48–49
 - preferred theme storage, 213–214
 - preferred themes, 210–211
 - searches, 50
 - table data information display, 222–223
 - theme page display/selections, 211–212
 - validation controls, 188–192
- user interface, elements, 10–16
- user profiles
 - data types, 172, 173–175
 - event/code tying, 180–183
 - information entry page, 176–188
 - information storage, 183–184, 187
 - phone numbers, 172
 - profile properties, 171
 - retrieving/editing, 184–187

user profiles (*continued*)

String data type, 172, 173–175

Textbox controls, 177–178

Web forms, 176–188

Web.config file, 173–175

ZIP codes, 172

user properties, editing/viewing,
284–287● **U** ●**validation controls**

CompareValidator, 191–192

CustomValidator, 192

properties, 188–192

RangeValidator, 190

RegularExpressionValidator,
190–191

RequiredFieldValidator, 189–190

ValidationSummary, 192

ValidationSummary control, failed
validations, 192**values**

Boolean data type, 228, 232

CompareValidator control, 191–192

CustomValidator control, 192

data-bound control display, 268–269GUID (Globally Unique Identifier),
232–234**primary key field**, 237–238, 241–244

RangeValidator control, 190

varchar (MAX) data type, SQL
Server, 230varchar data type, SQL Server, 230
View menu, user interface element,
13–14**views**

creating in Database Explorer, 249

data-bound control selections, 265

grid display control, 276–280

query results display, 250–251

saving, 251

SQL Server query, 248–254

switching between, 23, 30–31

user properties, 284–287

vw_aspnet_, 248

Visual Basic language

profile properties, 187–188

supported programming language, 21

Visual Studio, software-development
tool, 322**Visual Web Developer Express**

browser compatibility settings, 17–19

installation, 10

IntelliSense technology, 31

supported programming languages, 21

user interface elements, 10–16

vw_aspnet_ views, SQL Server, 248

● **W** ●W3C (World Wide Web Consortium),
HTML specifications, 327**WAT (Web Site Administration Tool)**

access rules, 45–48

membership sites, 40–43

roles (people categories), 43–45

user accounts, 48–51

Web publishing preparations,
332–333watermarks, background styling,
107–108

Web application, described, 321

Web browsers

compatibility issues, 17–19

viewing Web pages, 35–36, 65–66

Web forms. *See also* forms

Button control addition, 178–179

button addition, 178–179

Master Pages, 63–66

skin file creation, 205

Textbox controls, 177–178

- theme page display/selections, 211–212
- user profile information entry, 176–188
- uses, 75
- validation controls, 188–192
- Web pages. *See also* Master Pages
 - adding Master Pages to, 69–71
 - blank page creation, 75–77
 - bookmarks, 86
 - closing/opening, 23–24
 - code-behind files, 33–34
 - content page, 65
 - CSS class selectors, 117–118
 - CSS element selectors, 116–117
 - Default.aspx, 22–23
 - DIV styles, 120–121
 - editing techniques, 26–32
 - element class selectors, 119–120
 - eyebrow menu addition, 164–165
 - horizontal rules, 92
 - HTML Pages, 75
 - HTML tables, 77–84
 - hyperlinks, 84–87
 - Login controls, 130–131
 - Login.aspx, 135–136
 - navigation control addition, 155–158
 - object properties, 29–30
 - picture addition, 87–91
 - picture insertion, 28–29
 - quick links, 85–86
 - RecoverPassword.aspx, 144
 - saving changes before closing, 32–33
 - saving changes when closing, 24
 - server control addition, 125–130
 - site map addition, 158–164
 - Source view editing techniques, 92–96
 - style applications, 116–121
 - subtotal display, 314–318
 - switching between Design/Source views, 23
 - text addition, 27
 - text formatting, 27–28
 - text selections, 27–28
 - theme uses, 209–210
 - theme view/selections, 211–212
 - titles, 34–35
 - types, 75
 - undoing changes, 28
 - user profile information entry, 176–188
 - viewing in a browser, 35–36, 65–66
 - Web Forms, 75
 - Web User Control addition, 165–167
- Web presence provider, publishing to the Web, 19
- Web Site Administration Tool (WAT)
 - access rules, 45–48
 - membership sites, 40–43
 - roles (people categories), 43–45
 - user accounts, 48–51
 - Web publishing preparations, 332–333
- Web sites
 - ASP.NET forums, 329
 - ASP.NET QuickStart Tutorials, 17
 - ASP.NET specifications, 328
 - Cascading Style Sheets (CSS), 17
 - coolnerds.com, 16
 - creating new, 22–23
 - CSS (Cascading Style Sheets), 122, 328
 - domain names, 332
 - dotnetjunkies, 329
 - Google, 123
 - hosting services, 19
 - Microsoft technical communities, 329
 - MSDN (Microsoft Developer Network), 327
 - navigation considerations, 153–154
 - .NET Framework Developer Center, 17
 - online resources, 16–17
 - opening/closing, 37

Web sites (*continued*)

- renaming, 22
- site-wide default theme, 219–220
- SQL Server Developer Center, 17, 329
- Visual C# Developer Center, 17
- W3C (World Wide Web Consortium), 327
- World Wide Web Consortium, 122
- XHTML Home Page, 17
- XML (Extensible Markup Language), 17, 328
- Web User Controls, adding to a Web page, 165–167
- Web.config file
 - password constraints, 150–151
 - security trimming, 162
 - Solution Explorer display, 51–52
 - user profiles, 173–175
- Web.sitemap file
 - adding sitemap to a Web page, 158–161
 - control binding, 163–164
 - security trimming, 162–163
- white space
 - DataList control display, 299
 - styling, 108–110

Windows authentication, membership sites, 42–43

Windows, copying/pasting files to a folder, 26

wizards, Data Configuration, 262–273

World Wide Web Consortium (W3C), HTML specifications, 327

wraps, text around pictures, 89–90



XML (eXtensible Markup Language), specifications, 328



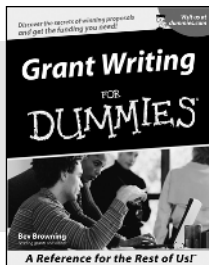
Z-indexes

absolute positioning, 194–195

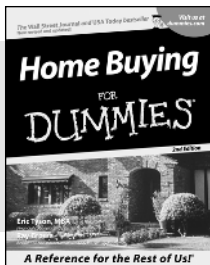
position styling, 111

ZIP codes, user profile element, 172

BUSINESS, CAREERS & PERSONAL FINANCE



0-7645-5307-0



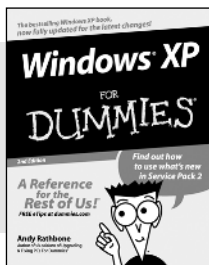
0-7645-5331-3 *†

Also available:

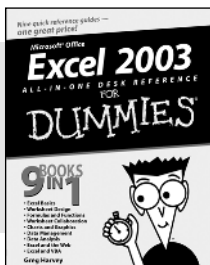
- ✓ Accounting For Dummies †
0-7645-5314-3
- ✓ Business Plans Kit For Dummies †
0-7645-5365-8
- ✓ Cover Letters For Dummies
0-7645-5224-4
- ✓ Frugal Living For Dummies
0-7645-5403-4
- ✓ Leadership For Dummies
0-7645-5176-0
- ✓ Managing For Dummies
0-7645-1771-6

- ✓ Marketing For Dummies
0-7645-5600-2
- ✓ Personal Finance For Dummies *
0-7645-2590-5
- ✓ Project Management For Dummies
0-7645-5283-X
- ✓ Resumes For Dummies †
0-7645-5471-9
- ✓ Selling For Dummies
0-7645-5363-1
- ✓ Small Business Kit For Dummies *†
0-7645-5093-4

HOME & BUSINESS COMPUTER BASICS



0-7645-4074-2



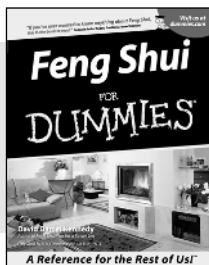
0-7645-3758-X

Also available:

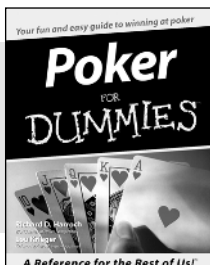
- ✓ ACT! 6 For Dummies
0-7645-2645-6
- ✓ iLife '04 All-in-One Desk Reference For Dummies
0-7645-7347-0
- ✓ iPAQ For Dummies
0-7645-6769-1
- ✓ Mac OS X Panther Timesaving Techniques For Dummies
0-7645-5812-9
- ✓ Macs For Dummies
0-7645-5656-8

- ✓ Microsoft Money 2004 For Dummies
0-7645-4195-1
- ✓ Office 2003 All-in-One Desk Reference For Dummies
0-7645-3883-7
- ✓ Outlook 2003 For Dummies
0-7645-3759-8
- ✓ PCs For Dummies
0-7645-4074-2
- ✓ TiVo For Dummies
0-7645-6923-6
- ✓ Upgrading and Fixing PCs For Dummies
0-7645-1665-5
- ✓ Windows XP Timesaving Techniques For Dummies
0-7645-3748-2

FOOD, HOME, GARDEN, HOBBIES, MUSIC & PETS



0-7645-5295-3



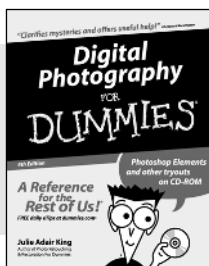
0-7645-5232-5

Also available:

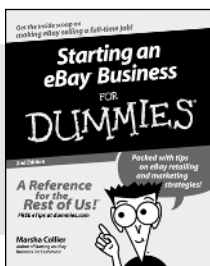
- ✓ Bass Guitar For Dummies
0-7645-2487-9
- ✓ Diabetes Cookbook For Dummies
0-7645-5230-9
- ✓ Gardening For Dummies *
0-7645-5130-2
- ✓ Guitar For Dummies
0-7645-5106-X
- ✓ Holiday Decorating For Dummies
0-7645-2570-0
- ✓ Home Improvement All-in-One For Dummies
0-7645-5680-0

- ✓ Knitting For Dummies
0-7645-5395-X
- ✓ Piano For Dummies
0-7645-5105-1
- ✓ Puppies For Dummies
0-7645-5255-4
- ✓ Scrapbooking For Dummies
0-7645-7208-3
- ✓ Senior Dogs For Dummies
0-7645-5818-8
- ✓ Singing For Dummies
0-7645-2475-5
- ✓ 30-Minute Meals For Dummies
0-7645-2589-1

INTERNET & DIGITAL MEDIA



0-7645-1664-7



0-7645-6924-4

Also available:

- ✓ 2005 Online Shopping Directory For Dummies
0-7645-7495-7
- ✓ CD & DVD Recording For Dummies
0-7645-5956-7
- ✓ eBay For Dummies
0-7645-5654-1
- ✓ Fighting Spam For Dummies
0-7645-5965-6
- ✓ Genealogy Online For Dummies
0-7645-5964-8
- ✓ Google For Dummies
0-7645-4420-9

- ✓ Home Recording For Musicians For Dummies
0-7645-1634-5
- ✓ The Internet For Dummies
0-7645-4173-0
- ✓ iPod & iTunes For Dummies
0-7645-7772-7
- ✓ Preventing Identity Theft For Dummies
0-7645-7336-5
- ✓ Pro Tools All-in-One Desk Reference For Dummies
0-7645-5714-9
- ✓ Roxio Easy Media Creator For Dummies
0-7645-7131-1

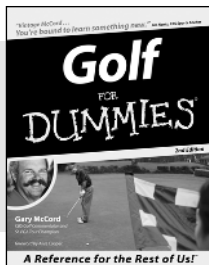
* Separate Canadian edition also available

† Separate U.K. edition also available

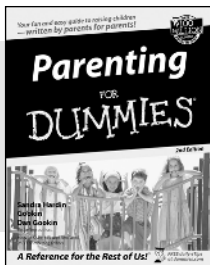
Available wherever books are sold. For more information or to order direct: U.S. customers visit www.dummies.com or call 1-877-762-2974. U.K. customers visit www.wiley.co.uk or call 0800 243407. Canadian customers visit www.wiley.ca or call 1-800-567-4797.



SPORTS, FITNESS, PARENTING, RELIGION & SPIRITUALITY



0-7645-5146-9



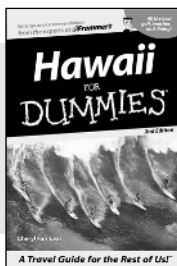
0-7645-5418-2

Also available:

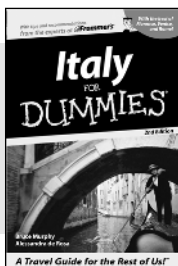
- ✓ Adoption For Dummies
0-7645-5488-3
- ✓ Basketball For Dummies
0-7645-5248-1
- ✓ The Bible For Dummies
0-7645-5296-1
- ✓ Buddhism For Dummies
0-7645-5359-3
- ✓ Catholicism For Dummies
0-7645-5391-7
- ✓ Hockey For Dummies
0-7645-5228-7

- ✓ Judaism For Dummies
0-7645-5299-6
- ✓ Martial Arts For Dummies
0-7645-5358-5
- ✓ Pilates For Dummies
0-7645-5397-6
- ✓ Religion For Dummies
0-7645-5264-3
- ✓ Teaching Kids to Read For Dummies
0-7645-4043-2
- ✓ Weight Training For Dummies
0-7645-5168-X
- ✓ Yoga For Dummies
0-7645-5117-5

TRAVEL



0-7645-5438-7



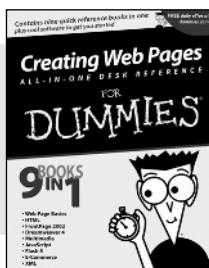
0-7645-5453-0

Also available:

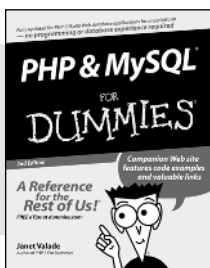
- ✓ Alaska For Dummies
0-7645-1761-9
- ✓ Arizona For Dummies
0-7645-6938-4
- ✓ Cancún and the Yucatán For Dummies
0-7645-2437-2
- ✓ Cruise Vacations For Dummies
0-7645-6941-4
- ✓ Europe For Dummies
0-7645-5456-5
- ✓ Ireland For Dummies
0-7645-5455-7

- ✓ Las Vegas For Dummies
0-7645-5448-4
- ✓ London For Dummies
0-7645-4277-X
- ✓ New York City For Dummies
0-7645-6945-7
- ✓ Paris For Dummies
0-7645-5494-8
- ✓ RV Vacations For Dummies
0-7645-5443-3
- ✓ Walt Disney World & Orlando For Dummies
0-7645-6943-0

GRAPHICS, DESIGN & WEB DEVELOPMENT



0-7645-4345-8



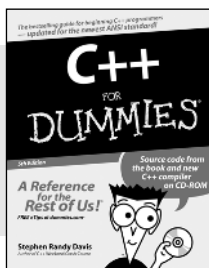
0-7645-5589-8

Also available:

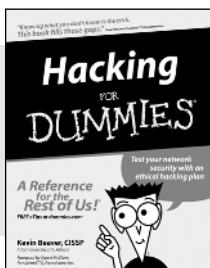
- ✓ Adobe Acrobat 6 PDF For Dummies
0-7645-3760-1
- ✓ Building a Web Site For Dummies
0-7645-7144-3
- ✓ Dreamweaver MX 2004 For Dummies
0-7645-4342-3
- ✓ FrontPage 2003 For Dummies
0-7645-3882-9
- ✓ HTML 4 For Dummies
0-7645-1995-6
- ✓ Illustrator cs For Dummies
0-7645-4084-X

- ✓ Macromedia Flash MX 2004 For Dummies
0-7645-4358-X
- ✓ Photoshop 7 All-in-One Desk Reference For Dummies
0-7645-1667-1
- ✓ Photoshop cs Timesaving Techniques For Dummies
0-7645-6782-9
- ✓ PHP 5 For Dummies
0-7645-4166-8
- ✓ PowerPoint 2003 For Dummies
0-7645-3908-6
- ✓ QuarkXPress 6 For Dummies
0-7645-2593-X

NETWORKING, SECURITY, PROGRAMMING & DATABASES



0-7645-6852-3



0-7645-5784-X

Also available:

- ✓ A+ Certification For Dummies
0-7645-4187-0
- ✓ Access 2003 All-in-One Desk Reference For Dummies
0-7645-3988-4
- ✓ Beginning Programming For Dummies
0-7645-4997-9
- ✓ C For Dummies
0-7645-7068-4
- ✓ Firewalls For Dummies
0-7645-4048-3
- ✓ Home Networking For Dummies
0-7645-42796

- ✓ Network Security For Dummies
0-7645-1679-5
- ✓ Networking For Dummies
0-7645-1677-9
- ✓ TCP/IP For Dummies
0-7645-1760-0
- ✓ VBA For Dummies
0-7645-3989-2
- ✓ Wireless All In-One Desk Reference For Dummies
0-7645-7496-5
- ✓ Wireless Home Networking For Dummies
0-7645-3910-8